PATENT ABSTRACTS OF JAPAN

(11)Publication number : **09-091185**
(43)Date of publication of application : **04.04.1997**

(51)Int.Cl. G06F 12/00
G06F 12/00
G06F 15/00
G06F 15/16

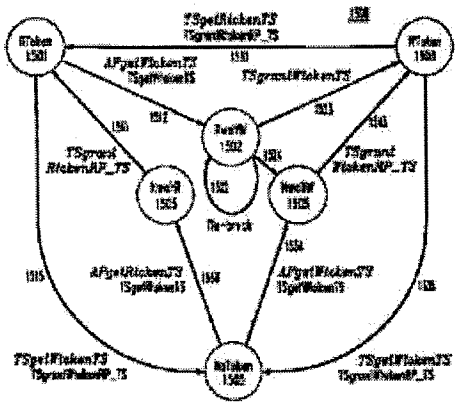
(21)Application number : **07-211279**
(22)Date of filing : **28.07.1995**

(71)Applicant : **AT & T CORP**
(72)Inventor : **RAO CHUNG-HWA HERMAN**
SKARRA ANDREA H

(30)Priority
Priority number : **94 282683** Priority date : **29.07.1994** Priority country : **US**

(54) DISTRIBUTED COMPUTING SYSTEM

(57)Abstract:
PROBLEM TO BE SOLVED: To provide a distributed system which permits plural copies of a file, but in which the meaning rule of a file system is a single copy of the file.
SOLUTION: Concerning copies of a file to be designated as 'copy file' in the distributed system, plural copies exist in the distributed system and the respective copies are set to be completely equal to the other copies. Namely, access to any copies results in the same as access when one local copy of the file exists and all access processes are executed on the same host. The synchronization of a write operation for the copy of the copy file is realized by transmitting a message designating the same operation to all the other element systems having the copy of the file whenever the element system executes an operation for changing the copy file to the copy of the copy file.



(19)日本国特許庁（ＪＰ）　(12) **公開特許公報**（Ａ）　(11)特許出願公開番号

特開平9－91185

(43)公開日　平成９年(1997)４月４日

(51)Int.Cl.[6]		識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ	12/00	５３５		Ｇ０６Ｆ	12/00	５３５Ｚ
		５４５				５４５Ａ
	15/00	３１０			15/00	３１０Ｕ
	15/16	３７０			15/16	３７０Ｎ

審査請求　未請求　請求項の数５　ＦＤ　（全 27 頁）

(21)出願番号　特願平7－211279

(22)出願日　平成７年(1995)７月28日

(31)優先権主張番号　２８２６８３

(32)優先日　1994年７月29日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390035493
エイ・ティ・アンド・ティ・コーポレーション
ＡＴ&Ｔ　ＣＯＲＰ.
アメリカ合衆国　10013－2412　ニューヨーク　ニューヨーク　アヴェニュー　オブ　ジ　アメリカズ　32

(72)発明者　チュン　ハー　ハーマン　ラオ
アメリカ合衆国，08820　ニュージャージー，エジソン，スプリングブルック　ドライブ　4304

(74)代理人　弁理士　三俣　弘文

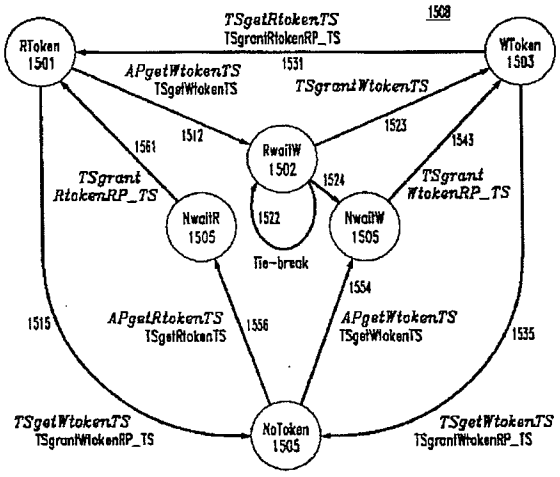
最終頁に続く

(54)【発明の名称】　分散コンピューティングシステム

(57)【要約】

【課題】　ファイルの複数のコピーを許容するがファイルシステムの意味規則はファイルの単一のコピーのものであるような分散システムを実現する。

【解決手段】　分散システムにおいて、「複製ファイル」として指定されるファイルは、分散システム内に複数のコピーが存在して、各コピーが他のコピーと完全に同等とされる。すなわち、いずれのコピーへのアクセスも、そのファイルのただ１つのローカルコピーが存在しすべてのアクセスプロセスが同一のホスト上で実行される場合のアクセスと同一の結果となる。複製ファイルのコピーに対する書き込み操作の同期は、要素システムが、複製ファイルを変更する操作を、複製ファイルのコピーに対して実行するときはいつでも、同一の操作を指定するメッセージを、その複製ファイルのコピーを有する他のすべての要素システムに送ることにより実現される。



1

【特許請求の範囲】

【請求項１】　分散データ処理システムの要素上で実行されるプロセスが資源に対して実行する操作を同期させる方法において、

前記プロセスにおいて、前記操作を実行するために要求されるトークンが前記要素内にあるかどうかを判断するステップと、

前記プロセスにおいて、前記トークンが前記要素内にある場合、前記トークンが利用可能になるまで待機して利用可能になると前記操作を実行し、前記トークンが前記要素内にない場合、前記要素内のトークンサーバが前記分散データ処理システム内の他の要素に前記トークンを要求するステップと、

前記プロセスにおいて、前記トークンが前記他の要素から到着したとき、前記トークンが利用可能になると前記操作を実行する操作実行ステップと、

前記トークンサーバにおいて、前記操作の完了後にはじめて前記他の要素への前記トークンに対する要求に応答するステップとからなることを特徴とする、分散データ処理システムの要素上で実行されるプロセスが資源に対して実行する操作を同期させる方法。

【請求項２】　前記操作実行ステップが、

前記プロセスが、前記操作によって要求される資源に対するロックを取得するまで前記プロセスを停止し、その後で前記操作を実行するステップと、

前記操作を実行した後に前記ロックを解放するステップとを有することを特徴とする請求項１の方法。

【請求項３】　前記操作が、前記他の要素において複製された、前記要素内のファイルに対する書き込み操作であることを特徴とする請求項２の方法。

【請求項４】　分散システム内に複製ファイルを生成する装置において、

前記複製ファイルの第１のコピーに対して操作を実行する第１ローカル操作実行手段と、

前記複製ファイルの第１のコピーに対して前記操作を実行する第２ローカル操作実行手段と、

各ローカル操作実行手段において、自己のファイルシステム手段で作成された複製ファイルのコピーに対する操作を他方のローカル操作実行手段に通知する操作通知手段と、

各ローカル操作実行手段において、他方のローカル操作実行手段内の操作通知手段に応答して、当該ローカル操作実行手段内の複製ファイルのコピーに対して、通知された操作を実行する通知操作実行手段と、

各ローカル操作実行手段において、複製ファイルのコピーに対する操作がどのコピーに対して実行されるかにかかわらず同じ結果を有するように複製ファイルのコピーに対する操作を同期させる手段とからなることを特徴とする、分散システム内に複製ファイルを生成する装置。

【請求項５】　複数の要素コンピューティングシステム

2

を有する分散コンピューティングシステムにおいて、

要素コンピューティングシステム内に配置され、相異なる要素コンピューティングシステム上にコピーを有する複製ファイルを、コピーに対して実行される操作がどのコピーに対して実行されるかにかかわらず同じ結果を有するように維持する分散手段からなることを特徴とする分散コンピューティングシステム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、コンピュータシステムに関し、特に、緩く接続された分散システムにおける動作の同期に関する。

【０００２】

【従来の技術】元来、コンピュータシステムは、単一のプロセッサと、ファイルを記憶するディスクドライブのような大容量記憶装置とから構成された。プロセッサのユーザはそれぞれプロセッサに接続された端末を有し、そのプロセッサを使用してファイルにアクセスすることができた。このようなシステムでは、すべてのアクセスはシステムの単一のプロセッサを通じてのものであり、システムには与えられたファイルの単一のコピーのみが存在した。プロセッサ、メモリ、および大容量記憶装置（例えばディスクドライブ）の価格が低下するにつれて、このようなシングルプロセッサシステムは分散システムによって置き換えられている。分散システムには、ネットワークへと接続されたいくつものローカルなプロセッサおよび記憶装置がある。このような分散システムの簡単な例としては、各ユーザがパーソナルコンピュータまたはワークステーションを有し、ワークステーションにファイルを提供するファイルサーバがあり、ワークステーションがローカルエリアネットワークによって相互におよびファイルサーバに接続されたものがある。もちろん、このような分散システムはそれ自体、より大きな分散システムの要素となることもあり、その場合には、他のシングルプロセッサシステムおよび分散システムが、ローカルエリアネットワークまたは電話システムのような広域ネットワークによって接続される。

【０００３】このような分散システムの利点には、ロバストネス（頑強さ）および速度が改善されることがある。ロバストネスの改善は、システムを構成する要素の数から生じる。例えば、１つのワークステーションが故障しても、ネットワーク上の他のワークステーションが役に立たなくなることはない。さらに、各ワークステーションは固有のファイルシステムを有し、そのため、システムにはファイルの複数のコピーが存在することがある。ファイルの１つのコピーが使用不能になった場合、他のワークステーションからコピーを使用可能である。速度の改善は、ユーザがローカルな処理能力および他人と共有していないローカルな記憶装置を有するということから生じる。

3

【０００４】ファイルのローカルなコピーは分散システムのロバストネスおよび速度を改善するが、書き込み可能なファイルのローカルなコピーは１つの主要な欠点を有する。それらのコピーの相互の一貫性を保つことが困難であるという欠点である。理想的には、分散システムにおけるファイル操作の意味規則（セマンティクス）は、プロセスのグループがプロセッサを共有するようなシステムにおけるものと同一であろう。例えば、ＵＮＩＸオペレーティングシステムでは（ＵＮＩＸはＸ／ＯＰＥＮ財団の商標である）、プロセスがデータをファイルに書き込むとき、他のプロセスは、最初のプロセスによる書き込みが完了するまで、そのファイルからデータを読み出すことも、そのファイルにデータを書き込むこともできない。これは、読み出しプロセスがファイルをオープンしたのが、書き込みプロセスがシステムコールを実行する前か後かにかかわらず成り立つ。

【０００５】利用可能な分散システムはローカルコピーの問題を「キャッシュ」によって処理している。サーバにファイルの単一の主コピー（マスタコピー）があり、そのファイルを使用しているワークステーションはそのファイルの一部または全部を含むキャッシュを有する。キャッシュされたコピーにおける変化はマスタコピーには即時には反映されず、その逆もそうであるため、キャッシュされたコピーは相互にあるいは主コピーと矛盾することがある。その結果、あるワークステーションによるファイルの読み出しが他のワークステーションによる書き込みの後にあっても、その読み出しは、キャッシュされたコピーを使用しているため、書き込みの結果を見ないことがある。

【０００６】キャッシュを使用するシステムのファイル操作の意味規則は一貫性の欠如を反映している。このようなシステムの一例は、アール．サンドバーグ(R. Sandberg)他、「Sun Network File Systemの設計と実装(Design and Implementation ofthe Sun Network File System)」、Proceedings of Summer Usenix（１９８５年６月）第１１９～１３０ページ、に記載されたSun Network File System（ＮＦＳ）である。ＮＦＳでは、与えられたファイルはファイルサーバに存在する。ワークステーションはその与えられたファイルのコピーを含むキャッシュを有することが可能である。キャッシュ内のコピーは、そのコピーがキャッシュにロードされた後３秒間はファイルサーバ内のコピーと同一であると仮定される。キャッシュされたディレクトリデータは、フェッチされた後３０秒間は有効であると仮定される。

【０００７】もう１つの例は、エム．カザー(M. Kazar)、「アンドリューファイルシステムにおける同期とキャッシュの問題(Synchronization and caching issues in theandrew file system)」、Proceedings of Winter Usenix（１９８８年）に記載されたアンドリューファイルシステムである。アンドリューファイルシステム

4

は、ファイルシステムコールが完了した後、そのファイルシステムコールの結果は、２つの重要な例外を除いて、ネットワークのどこでも即時に見えることを保証する。第１の例外は、書き込みシステムコールの場合、ファイルに書き込まれた新しいデータは、そのファイルに書き込みをしているプロセスがファイルを閉じるまでは、主コピーには実際には書き込まれないことである。第２の例外は、ファイルのキャッシュされたコピーは、プロセスがそのファイルをオープンするときに主コピーとの一貫性が検査されるだけである。従って、２つのプロセスがファイルを共有するとき、あるワークステーションで実行中の第１のプロセスは、他のワークステーションで実行中の第２のプロセスによって書き込まれたデータを、第２のプロセスがそのファイルをクローズした後に第１のプロセスがそれをオープンするまでは、見ないことになる。

【０００８】

【発明が解決しようとする課題】ファイルの複数のコピーを許容するがファイルシステムの意味規則はファイルの単一のコピーのものであるような分散システムが必要とされている。本発明の目的は、このような分散システムを実現することである。

【０００９】

【課題を解決するための手段】本発明の分散システムでは、いくつかのファイルが「複製ファイル」として指定される。複製ファイルは、分散システム内に複数のコピーが存在して、各コピーが他のコピーと完全に同等であるようなファイルである。ファイルが完全に同等であるとは、いずれのコピーへのアクセスも、そのファイルのただ１つのローカルコピーのみが存在しすべてのアクセスプロセスが同一のホスト上で実行される場合のアクセスと同一の結果となることである。こうして、本発明の分散システムは、プロセスが同一のホストで実行されるようなシステム上の単一のファイルへのアクセスのファイルシステム意味規則と、ファイルのローカルコピーとの利点を合わせ持っている。

【００１０】本発明のもう１つの特徴は、複製ファイルのコピーに対する書き込み操作を同期する技術にある。分散システムホストの要素システムが、複製ファイルを変更する操作を、複製ファイルのコピーに対して実行するときはいつでも、同一の操作を指定するメッセージが、その複製ファイルのコピーを有する他のすべての要素システムに送られる。すべてのメッセージは同一のチャネルを通じて移動するため、操作の順序はすべてのコピーに対して同一である。書き込み操作は、基本書き込み操作であることも可能であり、あるいは、複製ファイルに含まれるデータに関する操作の指定であることも可能である。

【００１１】もう１つの特徴は、複製ファイルに対する順次読み出し操作が書き込み操作と同期していることで

5

あり、その結果、シングルプロセッサシステムにおける通常のファイルに対して実行される読み出し操作と同一の意味規則を有することになる。

【００１２】順次読み出し操作と書き込み操作の同期は、各複製ファイルに対する読み出しトークンおよび書き込みトークンと、分散システムの各要素内のトークンサーバとを使用する分散同期システムによって実現される。複製ファイルに対して書き込み操作を実行するシステムの要素は書き込みトークンを有していなければならず、順次読み出し操作を実行する要素は、読み出しトークンまたは書き込みトークンを有していなければならない。要素がトークンを取得するには、その要素のトークンサーバが他の要素のトークンサーバにそのトークンを要求する。トークンサーバは、未完了の書き込み操作を指定するすべてのメッセージを他の要素から受信した後にのみそのトークンを受信する。その結果、書き込み操作はすべてのローカルコピーに対して同じ順序で実行され、順次読み出し操作はすべてのコピーに対して同じ結果を有する。

【００１３】本発明の同期システムのもう１つの特徴は、複製ファイルの各ローカルコピーに対する７個のロックを使用して実装されることである。これらのロックは、標準的なオペレーティングシステムの共有ロックおよび排他ロックを使用して順に実装される。

【００１４】本発明のさらにもう１つの特徴は、複製ファイルへのアクセスのトランザクション的同期をサポートするために使用することができることである。

【００１５】本発明の利点は、複製ファイルを保守するために必要な操作が、分散システムのユーザレベルで実装可能であることにある。その結果、本発明は、特殊なハードウェアや特殊なオペレーティングシステムを必要としない。好ましい実施の形態は、ユーザレベルのバックアップファイルシステムの変更として実装される。

【００１６】

【発明の実施の形態】

［ライブラリを使用したインタフェースの変更：図２］既に説明したように、コンピュータシステムは層化されている。各層は隣の上位層へのインタフェースを提供する。上位層は、下位層のインタフェースによって要求されるように下位層が実行する操作を指定する。上位層が下位層によって要求されるインタフェースに従っていない場合、上位層と下位層の間にアダプタ層を追加しなければならない。アダプタ層の目的は、上位層によって期待されるインタフェースに従ってなされる操作指定を、下位層のインタフェースによって要求される操作指定に変換することである。この技術を使用して、例えば、ＭＳＤＯＳオペレーティングシステムを実行しているＰＣが、ユーザには、ＵＮＩＸオペレーティングシステムを実行しているコンピュータであるかのように見えるようにすることが可能である。

6

【００１７】アダプタ層が多くのアプリケーションプログラムによって要求されるとき、これはライブラリルーチンのセットとして実装されることが多い。その名前からわかるように、ライブラリルーチンは、コンピュータシステムのサブシステムのメーカが、そのコンピュータシステムのユーザに対して、アプリケーションプログラムとともに使用するように提供するルーチンのことである。図２に、どのようにしてライブラリルーチンがアダプタ層をなすように使用されるかを示す。ユーザプログラム２０１は、次の層（この場合には、システムルーチンのセット）へのインタフェース２０６を有する。しかし、ユーザプログラム２０１が使用されるコンピュータシステムのシステムルーチンは、インタフェース２１３を有する。インタフェース２０６とインタフェース２１３の相違は、図２では、インタフェースを表す線の形の相違によって表されている。アダプタ層はライブラリルーチン２０７からなり、ユーザプログラム２０１によって要求される隣の上位層に対するインタフェース２０６と、システムルーチン２０５によって要求される隣の下位層に対するインタフェース２１３とを有する。インタフェースは実際にはファンクション（関数）呼出しからなり、ライブラリルーチン２０７内のルーチンは、インタフェース２０６によって要求されるファンクション呼出し２０３に応答して、インタフェース２１３によって要求されるファンクション呼出しを生成し、ファンクション呼出し２０３によって指定される操作を実行することによって動作する。システムルーチン２１５は、終了すると、その実行の結果を矢印２１１で示されるようにライブラリルーチン２０７に返し、続いてライブラリルーチン２１１はその結果を復帰２０５によって示されるようにユーザプログラム２０１に返す。

【００１８】［動的リンクライブラリルーチンを使用したインタフェースの再定義］インタフェースを再定義するためのライブラリルーチンの有用性は、従来のシステムでは、ユーザプログラム２０１に対する実行可能コードが生成されるときにユーザプログラム２０１にリンクされなければならないということによって制限されていた。この場合のリンクとは、ユーザプログラム２０１におけるライブラリルーチンの呼出しが、ライブラリルーチン２０７のコピー内のライブラリルーチンの位置に関係づけられるプロセスをいう。リンクは実行可能コードが生成されるときに行われなければならなかったため、実行可能コードのコピーしか有しないユーザは、あるライブラリルーチン２０７の他のライブラリルーチン２０７のセットと置換することは不可能であった。

【００１９】現在ではコンピュータシステムは発展してきており、ライブラリルーチンはユーザプログラムに動的にリンクすることが可能である。このようなコンピュータシステムでは、リンクは、ユーザプログラムを実行するプロセスが実行前にコンピュータシステムのメモリ

にロードされるときに行われる。動的リンクにより、ユーザプログラムのオブジェクトコードを変更せずに、あるライブラリルーチンのセットを他のセットと置換することが可能であり、それによって、ユーザプログラムが動作するシステムの挙動を変えることが可能である。動的リンクについての説明は、「共有ライブラリ(Shared Libraries)」、Sun Microsystems, Inc.、米国カリフォルニア州マウンテン・ビュー（１９８８年5月）、に記載されている。

【００２０】図3に、どのようにして動的リンクを使用してシステムの挙動を変えるかを示す。システム１（３０１）において、ユーザプロセス３０６はアプリケーションプログラム３０９を実行しており、これに、オペレーティングシステムライブラリ１（３１５）が動的にバインドされている。オペレーティングシステムライブラリ１（３１５）は、コール３１１および復帰３１３によって示されるアプリケーションプログラム３０９へのインタフェースを提供し、カーネルサーバ３０５へのコール３１７およびカーネルサーバ３０５からの復帰３１９を使用してコール３１１によって指定される操作を実行する。システム２では、ユーザプロセス３０６は同じアプリケーションプログラム３０９を実行し同じカーネルサーバ３０５を使用しているが、こちらでは、オペレーティングシステムライブラリ２（３２１）によってオペレーティングシステムライブラリ１（３１５）が置換されている。オペレーティングシステムライブラリ２（３２１）は、オペレーティングシステムライブラリ１（３１５）が行うすべてのことを行う。すなわち、システム３０１を、システム３０１のように挙動するが副次的効果３２３も生成するシステム３０３へと変換するのに必要なことは、オペレーティングシステムライブラリ２（３２１）をオペレーティングシステムライブラリ１（３１５）の代わりにユーザプログラム３０９に動的にリンクすることだけである。

【００２１】［動的リンクライブラリを使用したユーザレベル名前空間の作成］さらに、図4に、どのようにして動的リンクオペレーティングシステムライブラリ４０３を使用してユーザレベル名前空間４０５を作成し、どのようにしてユーザレベル名前空間４０５を使用して副次的効果３２３を制御するかを示す。ファンクション、ファイルおよびデバイスのようなコンピュータシステムにおけるエンティティはプログラムにおいて名前によって呼ばれ、プログラムで使用される名前をその名前によって表されるエンティティに関係づけることはコンピュータシステムの名前空間の機能である。従来のコンピュータシステムでは、ユーザプログラムによって使用される名前空間はオペレーティングシステムによって作成され保守されている。システム４０１では、オペレーティングシステムライブラリ４０３がユーザプロセス４０９に対する１つ以上のユーザレベル名前空間４０５を作成

し保守する。ユーザレベル名前空間４０５がライブラリルーチン４０３によって使用されることを可能にする１つの方法は、カーネルサーバ３０５によってユーザプログラム３０９に提供されるファイルシステムとは挙動、構造、またはその両方において異なるユーザレベルの論理ファイルシステムを作成することである。その後、この論理ファイルシステムを使用して、副次的効果３２３を制御する。例えば、システム４０１がバックアップファイルシステムである場合、副次的効果３２３は、バックアップファイルシステムを生成するために要求されるものであり、ユーザレベル名前空間４０５は、カーネルサーバ３０５によって提供されるファイルシステムにおけるどのファイルがバックアップファイルシステムにバックアップされるべきかを指定することが可能である。図４から明らかなように、ユーザレベル名前空間４０５はユーザプロセス４０９の環境の一部である。

【００２２】［ユーザレベルバックアップファイルシステムの概観：図５～図６］上記の動的リンクライブラリおよびユーザレベル名前空間を使用して、アプリケーションプログラムを実行しているアプリケーションプロセスによって変更されたファイルのうち選択したもののみを自動的にバックアップするユーザレベルバックアップファイルシステムを形成することが可能である。図５に、そのようなユーザレベルバックアップファイルシステム５０１を示す。システム５０１は、２つのコンピュータシステムによって実装される。主システム５１１では、アプリケーションプロセス５０３が実行され、バックアップシステム５１３では、アプリケーションプロセス５０３によって変更されたファイルのバックアップコピーが保守される。主システム５１１およびバックアップシステム５１３は通信媒体によって接続され、これによって、主システム５１１で実行されているプロセスからのメッセージをバックアップシステム５１３で実行されているプロセスへ送ることができる。

【００２３】主システム５１１上のシステム５０１の要素は、アプリケーションプロセス５０３およびカーネルサーバ３０５（ａ）である。カーネルサーバ３０５（ａ）は主システム５１１にファイルシステムを提供する。図５において、ファイルシステムは主システム５１１に対してローカルなディスク３０７（ａ）によって表されているが、これは他のシステム上に位置するリモートファイルシステムでも全くかまわない。いずれの場合にも、カーネルサーバ３０５（ａ）は、アプリケーションプロセス５０３からのコール３１７に応答して、提供するファイルシステムに対するファイル操作を実行し、結果３１９をプロセス５０３に返し、自分自身必要な操作をディスク３０７（ａ）に対して実行する。アプリケーションプロセス５０３は、動的リンク可能ライブラリを使用して、カーネルサーバ３０５（ａ）とともにファイル操作を実行する。主システム５１１では、このライ

ブラリは、ｌｉｂ．３ｄ（５０７）と呼ばれる新たなライブラリによって置換されている。ライブラリ５０７は、いくつかのファイルを変更するファイル操作を指定するコール３１１に応答して、カーネルサーバ３０５への適当なコール３１７を提供するだけでなく、バックアップメッセージ５１２をバックアップシステム５１３に送る。変更の結果バックアップメッセージ５１２を送ることになるファイルはフロントエンド複製ツリー（ＦＲＴ）５０５で指定される。複製ツリー５０５は、矢印５０６で示されるように、ｌｉｂ．３ｄ（５０７）内のルーチンによって保守され使用される。このようにして、複製ツリー５０５は、変更の結果システム５１３上のバックアップファイルを変更することになるファイルからなるユーザレベル論理ファイルシステムを定義する。

【００２４】バックアップシステム５１３上のシステム５０１の要素は、バックエンドサーバ５１５、ユーザレベルプロセス、ならびにカーネルサーバ３０５（ｂ）およびディスク３０７（ｂ）であり、バックアップシステム５１３のための標準的なファイルシステムサーバおよびディスクドライブである。カーネルサーバ３０５（ｂ）はバックエンドサーバ５１５にファイルシステムを提供する。図５では、ファイルシステムのためのデータはローカルディスク３０７（ｂ）上に記憶されている。しかし、これはリモートシステムに記憶することも可能である。バックエンドサーバ５１５は、カーネルサーバ３０５（ｂ）へのコール３１７によってファイル操作を実行し、そのコールの結果をサーバ３０５（ｂ）から受け取る。バックエンドサーバ５１５はバックエンドマップ５１７を保守する。バックエンドマップ５１７は、フロントエンド複製ツリー５０５によって指定されるファイルを、バックアップとして使用されるバックアップシステム５１３のファイルシステム内のファイル上にマップする。カーネルサーバ３０５（ａ）によって生成されるファイルシステムとカーネルサーバ３０５（ｂ）によって生成されるファイルシステムが同一の名前空間を有するような実施の形態では、バックエンドマップ５１７は不要となる。

【００２５】どのようにしてシステム５０１が動作するかは図６から明らかとなる。図６には、ファイルを変更するライブラリ５０７内のルーチン６０１の形式の一般的概略が示されている。ルーチン名６０３およびこのルーチンがとる引数６０５は、ライブラリ５０７によって置換されたライブラリ内のファイル操作を実行するために使用される関数の名前および引数と同一である。その結果、アプリケーションプログラム５０９におけるこのルーチンの呼出しはルーチン６０１を呼び出す。必要な準備を実行した後、ルーチン６０１はカーネルサーバ３０５（ａ）に、ルーチン６０１によって置換されたルーチンと同じファイル操作を実行させる。この操作が成功した場合、ルーチン６１３は、変更されたファイルの名前とともに関数６１３を呼び出し、変更されたファイルがバックアップされるべきであるということをフロントエンド複製ツリー５０５が示しているかどうかを判定する。フロントエンド複製ツリーがそのように示している場合、関数６１５は引数６１７によりメッセージ５１２をバックアップシステム５１３へ送る。メッセージ５１２は、バックアップシステム５１３が、サーバ３０５（ａ）によって提供されるファイルシステム上でちょうど実行されたのと全く同じ操作をバックアップファイルシステムに対して実行するよう要求する。このメッセージを送った後、ルーチン６０１は復帰する。これは、ファイルがフロントエンド複製ツリー５０５内になかった場合、または、関数６０７によって指定される操作が成功しなかった場合も同様である。図６で６１１とラベルされているコードのセクションは、副次的効果（この場合はメッセージ５１２）を指定する。ここで注意すべきルーチン６０１の特徴は、メッセージ５１２がバックアップシステム５１３に送られるのはファイル操作が主システム５１１で成功した場合のみであるということである。これは、不成功の操作はバックアップする必要がないためである。

【００２６】システム５０１には一般的に２つのクラスのファイル操作がある。フロントエンド複製ツリー５０５およびバックエンドマップ５１７によって実装されたユーザレベル名前空間４０５を変更するものとそうでないものである。第２のクラスの操作の一例は、フロントエンド複製ツリー５０５に指定されたファイルへの書き込みである。ｌｉｂ．３ｄ（５０７）内の書き込みファンクションは、ｌｉｂ．３ｄによって置換されたライブラリ内の書き込みファンクションと同じインタフェースを有する。好ましい実施の形態では、これは、引数として、ファイルを指定するためにカーネルサーバ３０５（ａ）によって使用される整数のファイルディスクリプタと、書き込むデータを含むバッファへのポインタと、書き込むデータのサイズを示す整数とをとる。ｌｉｂ．３ｄ内の書き込みファンクションは、カーネルサーバ３０５（ａ）が、ファイルディスクリプタによって指定されるファイルに対してシステム書き込みファンクションを実行することを要求し、その操作が成功した場合、このファンクションは、そのファイルディスクリプタによって指定されるファイルがフロントエンド複製ツリー５０５内に存在するかどうかを検査する。存在する場合、ファンクションはバックアップシステム５１３内のバックエンドサーバ５１５へ書き込みメッセージ５１２を送り復帰する。このメッセージは、カーネルサーバ３０５（ａ）によってちょうど書き込まれたファイルを指定し、カーネルサーバ３０５（ａ）によって提供されるファイルシステム内のシステム書き込み操作によってちょうど実行された書き込み操作を全く同様にバックアップファイルシステムにおいて実行するのに必要な情報を含

む。バックエンドサーバ５１５は、このメッセージを受け取ると、バックエンドマップ５１７を使用して、カーネルサーバ３０５（ｂ）がバックアップファイルに対して使用するファイルディスクリプタを判定してから、カーネルサーバ３０５（ｂ）によって提供されるシステム書き込みファンクションを使用して、このメッセージによって提供されるデータおよび位置の情報を用いてバックアップファイルに対して書き込み操作を実行する。

【００２７】ユーザレベル名前空間４０５を変更する操作の簡単な場合はファイル削除である。ｌｉｂ．３ｄによって提供される削除ファンクションは、まずカーネルサーバ３０５（ａ）にファイルを削除するよう要求する。この削除が終了すると、削除ファンクションは、削除されたファイルに関する情報をフロントエンド複製ツリー５０５から削除することが必要かどうかを検査する。それが必要な場合、ファンクションはその情報を削除する。次に、ファンクションは、削除に必要なメッセージをバックエンドサーバ５１５へ送り復帰する。バックエンドサーバ５１５は、このメッセージを受け取ると、バックエンドマップ５１７内でそのファイルを見つけ、カーネルサーバ３０５（ｂ）にそのファイルを削除するよう要求するとともに、この削除によって要求される操作をバックエンドマップ５１７に対して実行する。

【００２８】より複雑な例は名前変更である。カーネルサーバ３０５（ａ）によって提供されるファイルシステム内のファイルの名前変更がユーザレベル名前空間４０５において引き起こす結果には３つの可能性がある。

【００２９】１．そのファイルの古い名前がユーザレベル名前空間４０５の一部であり、新しい名前もまたユーザレベル名前空間４０５の一部である場合、そのファイルはユーザレベル名前空間４０５内にとどまる。

２．そのファイルの古い名前はユーザレベル名前空間４０５の一部でないが、新しい名前はユーザレベル名前空間４０５の一部である場合、そのファイルはユーザレベル名前空間４０５に追加される。

３．そのファイルの古い名前はユーザレベル名前空間４０５の一部であるが、新しい名前はユーザレベル名前空間４０５の一部でない場合、そのファイルはユーザレベル名前空間４０５から削除される。

【００３０】第１の場合、ｌｉｂ．３ｄの名前変更ファンクションは、カーネルサーバ３０５（ａ）に、そのファイルシステムにおける名前変更を行うよう要求する。次に、このファンクションは、名前変更されたファイルがユーザレベル名前空間４０５内にあるかどうかを検査し、ユーザレベル名前空間４０５内にある場合、名前変更ファンクションは、その変更を反映するようにフロントエンド複製ツリー５０５を変更し、バックエンドサーバ５１５における名前変更を要求するメッセージをバックエンドサーバ５１５へ送り、復帰する。このメッセージはもちろん、旧パス名および新パス名を含む。バックエンドサーバ５１５は、このメッセージを受信すると、カーネルサーバ３０５（ｂ）に名前変更を要求する。

【００３１】第２の場合、名前変更ファンクションは、サーバ３０５（ａ）に名前変更を要求し、前のように、名前変更されたファイルがユーザレベル名前空間４０５内にあるかどうかを検査するが、今度は、ファンクションは、名前変更されたファイルをフロントエンド複製ツリー５０５から削除し、メッセージをバックエンドサーバ５１５へ送り、復帰する。バックエンドサーバ５１５へのメッセージは、そのファイルに対する削除メッセージである。このメッセージに応答して、バックエンドサーバ５１５はカーネルサーバ３０５（ｂ）にバックアップファイルを削除させる。

【００３２】第３の場合も、前のように、名前変更ファンクションは名前変更を要求するが、今度は、２つのメッセージを送らなければならない。第１のメッセージは、ユーザレベル名前空間４０５へ移動されたファイルの名前を有するファイルをバックアップシステム５１３内に作成することを要求する。バックエンドサーバ５１５はこのメッセージに応答してカーネルサーバ３０５（ｂ）がそのファイルを作成することを要求し、バックエンドマップ５１７内にそのファイルのエントリを作成する。その後、名前変更ファンクションはユーザレベル名前空間４０５に移動されたファイルの現在の内容とともに書き込みメッセージを送る。バックエンドサーバ５１５はこの書き込みメッセージに応答して、カーネルサーバ３０５（ｂ）によって、バックアップシステム５１３内のバックアップファイルにその内容を書き込む。

【００３３】以上のことからわかるように、主システム５１１内のカーネルサーバ３０５（ａ）によって実行される単一の操作は、バックエンドサーバ５０５がカーネルサーバ３０５（ｂ）に一連の操作を実行させることを要求する。さらに理解されるように、ｌｉｂ．３ｄ（５０７）内のファンクションによって実行される操作の最後には、バックエンドマップ５１７およびフロントエンド複製ツリー５０５は常に同じ状態になる。

【００３４】［好ましい実施の形態の実装：図７～図１１］図７に、ユーザレベルバックアップファイルシステムの好ましい実施の形態７０１の詳細ブロック図を示す。この好ましい実施の形態は、一方のプロセッサがＵＮＩＸオペレーティングシステムのSunOS４．１バージョンを実行しており他方のプロセッサがＵＮＩＸオペレーティングシステムのＭＩＰＳ４．５バージョンを実行しているシステムにおいて実装された。システム７０１には要素の２つのグループがある。一方のグループの要素はバックアップファイル操作を実行し、他方のグループの要素はシステム７０１をフォールトトレラントにする。以下の説明では、まず、バックアップファイル操作を実行する要素について説明し、その後で、フォールトトレランスを提供する要素について説明する。

13

【００３５】主システム５１１から始めると、アプリケーションプロセス５０３は、アプリケーションプログラム５０９、動的リンク可能ライブラリｌｉｂ．３ｄ（５０７）、およびフロントエンド複製ツリー５０５を有する。ライブラリ５０７のファンクションはファイル操作の副次的効果としてバックアップファイル操作を実行する。システム５０１において、ファイル操作は、カーネルサーバ３０５（ａ）によって実行される。ライブラリ５０７内のファンクションによって生成されるメッセージは、パイプ７１０によってバックアップシステム５１３へ運ばれる。パイプ７１０は、パイププロセス７１１によってアプリケーションプロセス５０３に提供され、パイププロセス７１１自体、パイプ７０９によってアプリケーションプロセス５０３と通信する。以下でさらに詳細に説明するように、パイププロセス７１１は、バックアップシステム５１３上にバックアップを作成するすべてのアプリケーションプロセス５０３によって使用される単一のパイプ７１０を提供する。

【００３６】次に、好ましい実施の形態におけるバックアップシステム５１３において、バックエンドサーバ５１５は２つのプロセス、すなわち、バックエンドログプロセス（ＢＬＰ）７１６およびシステムコールエンジン（ＳＹＳＣＡＬＬ　ＥＮＧ）７１５に分かれる。いずれもカーネルサーバ３０５（ｂ）を使用してファイル操作を実行する。バックアップファイルに加えて、カーネルサーバ３０５（ｂ）によって保守されるファイルシステムはログファイル７０３（ｂ）を含む

【００３７】動作は以下の通りである。アプリケーションプロセス５０３は、初期化されると、パイプ７１０を指定するファイル識別子をパイププロセス７１１から取得する。アプリケーションプログラム５０９の実行の結果、ファイル操作が実行されると、ｌｉｂ．３ｄ（５０７）内のその操作に対するファンクションは、カーネルサーバ３０５（ａ）によって提供されるファイルシステムに対して、カーネルサーバ３０５（ａ）にそのファンクションを実行させ、さらに、パイプ７１０を通じてメッセージをバックアップシステム５１３に送る。このメッセージは、バックアップシステム５１３に到着すると、バックアップログプロセス７１６によって受け取られる。バックアップログプロセス７１６は、カーネルサーバ３０５（ｂ）によって提供されるファイルシステム内のログファイル７０３（ｂ）内にそのメッセージをログする。ログファイル７０３（ｂ）がメッセージを有するときにはいつでも、そのメッセージは、到着順に、システムコールエンジンプロセス７１５によって読み出される。好ましい実施の形態では、バックエンドマップ５１７はシステムコールエンジンプロセス７１５に属する。システムコールエンジンプロセス７１５は、メッセージを読み出すと、カーネルサーバ３０５（ｂ）に、そのメッセージによって要求されるファイル操作を実行させ、システムコールエンジンプロセス７１５自信は、そのメッセージによって要求されるようにバックエンドマップ５１７を保守する。

14

【００３８】［システム７０１のフォールトトレラント動作］システムのフォールトトレラント動作には、故障が検出され、検出された故障に応じてシステムが動作を継続することができるようになっていることが要求される。好ましい実施の形態では、故障の検出およびその故障への応答は、ＷａｔｃｈＤという、分散システムをフォールトトレラントにするためのユーザレベルのシステムによって扱われる。ＷａｔｃｈＤについての詳細は、ワイ．フアン(Y. Huang)、シー．キンタラ(C. Kintala)、「ソフトウェア実装フォールトトレラント：技術と経験(Software Implemented Fault Tolerance: Technologies andExperiences)」、第２３回フォールトトレラントコンピューティングに関する国際会議(23rd International Conference on Fault Tolerant Computing)、フランス国ツールーズ、１９９３年6月２２～２４日、に記載され、また、米国特許出願第０７／９５４，５４９号（発明者：ワイ．フアン(Y. Huang)、出願日：１９９２年9月３０日）の主題ともなっている。本発明の説明のためには、ＷａｔｃｈＤシステムが、ｌｉｂｆｔというライブラリと、分散システムの各ノード上の１つのモニタプロセスとを含むことを理解していればよい。ｌｉｂｆｔは、ＷａｔｃｈＤにプロセスを登録する操作、自動バックアップ用にメモリの領域を指定する操作、および、そのメモリ領域に対してチェックポイント操作を実行する操作などを実行するルーチンを含む。モニタプロセスは、ＷａｔｃｈＤに登録されたユーザプロセスをモニタするとともに、相互をモニタする。モニタは、登録されているプロセスが故障したと判定すると、そのプロセスを再起動する。プロセスは、ｌｉｂｆｔファンクションによって再起動されたときに何が起きたかを判定することが可能である。分散システムの１つのノード上のユーザプロセスをモニタする間、モニタは、重要データ（これもまたｌｉｂｆｔファンクションを使用して定義される）のコピーを分散システムの他のノードへ移動することが可能である。そのモニタのノードが故障すると、他のノード上のモニタがその故障を検出し、重要データの現在のコピーを使用して当該他のノード上でユーザプロセスを再起動する。故障したノードが復旧すると、そのノードのモニタは、他のノードからの重要情報を使用してユーザプロセスを再起動し、ユーザプロセスが再起動されたことを示すメッセージを送る。他のノードのモニタは、そのメッセージを受け取ると、当該他のノードで実行されているユーザプロセスを終了する。一般に、ＷａｔｃｈＤモニタはリング構成で配置され、各モニタはリングにおける隣のモニタをモニタする。リング内のノードの数およびユーザプロセスの重要データのコピーを受け取るモニタの数は、ＷａｔｃｈＤに登録さ

15

れたユーザプロセスを再起動することができなくなる前に分散システムのいくつのノードが故障しなければならないかを決定する。

【００３９】好ましい実施の形態では、主システム５１１およびバックアップシステム５１３はそれぞれＷａｔｃｈＤモニタを有する。これらのモニタとシステム７０１の要素の間の関係は、破線矢印７２１で示されている。主システム５１１のモニタはモニタ７１７である。破線矢印７２１で示されるように、モニタ７１７は、パイププロセス７１１、フロントエンドログプロセス７０５、およびシステム５１３内のモニタ７１９を監視する。モニタ７１９は、モニタ７１７、システムコールエンジンプロセス７１５、およびバックエンドログプロセス７１６を監視する。

【００４０】図７に示されるように、システム７０１は、フロントエンドログプロセス７０５、パイププロセス７１１、システムコールエンジン７１５、バックエンドログプロセス７１６における故障、およびシステム５１３の故障を処理することができる。この設計は、フォールトトレランスを与えるシステム７０１の２つの部分を有し、２つの主要な目的を有する。

【００４１】・パフォーマンスに関して、回復のオーバヘッドが少ないことを保証する。

・故障および回復がアプリケーションに透過的であり、実行中のアプリケーションが停止しないことを保証する。

【００４２】回復手続きは、ＷａｔｃｈＤがシステムにおける最も信頼性のある要素であるという仮定に基づく。その理由は、ＷａｔｃｈＤは非常に単純なタスクを実行し、故障後に自己回復が可能であるためである。

【００４３】以下では、バックアップシステム５１３の故障からの回復について詳細に説明し、他のプロセスの故障からの回復についても概観する。バックアップシステム５１３の故障から始めると、このような場合、システム７０１は以下のように動作する。モニタ７１７は、システム５１３の故障を検出すると、パイププロセス７１１に通知する。パイププロセス７１１はフロントエンドログプロセス７０５を作成し、パイプ７１０のファイルディスクリプタをフロントエンドログプロセス７０５へのパイプ７０７のファイルディスクリプタで置換する。アプリケーションプロセス５０３によって使用されるメッセージファンクションは、パイプ７１０の故障を検出すると、パイププロセス７１１にパイプの新しいファイルディスクリプタを要求する。パイププロセス７１１は、フロントエンドログプロセス７０５に接続されたパイプ７０７のファイルディスクリプタをそのメッセージファンクションに与え、メッセージファンクションによって送られたメッセージは、バックエンドログプロセス７１６ではなくフロントエンドログプロセス７０５へ行く。フロントエンドログプロセス７０５は、そのメッセージを受け取ると、そのメッセージを主システム５１１内のログファイル７０３（ａ）に入れる。

16

【００４４】好ましい実施の形態では、メッセージファンクションはパイプ７１０の故障を以下のように検出する。プロセス５０３はＴＣＰ／ＩＰプロトコルを使用してパイプ７１０を通じてメッセージを送る。このプロトコルでは、前のメッセージが受け取られた場合に限り次のメッセージを送ることができる。従って、ライブラリルーチン５０７内のファンクションによって使用されるメッセージファンクションは、２つのメッセージ、すなわち、実際のメッセージおよびダミーのメッセージを送ることによってパイプ７１０を通じてメッセージを送る。メッセージファンクションがダミーメッセージを送ることができる場合、実際のメッセージは到着したことになる。システム５１３が故障すると、パイプ７１０を通じて送られたメッセージは到着せず、ダミーメッセージを送ることはできない。

【００４５】バックアップファイルシステム５１３が回復すると、モニタ７１９は、システムコールエンジン７１５およびバックエンドログプロセス７１６を再起動し、モニタ７１７に通知する。モニタ７１７はパイププロセス７１１に通知し、パイププロセス７１１は、パイプ７１０のファイルディスクリプタを取得してフロントエンドログプロセス７０５を終了させる。バックエンドログプロセス７１６は、システム５１３において再起動されると、カーネルサーバ３０５（ａ）からログファイル７０３（ａ）のコピーを取得し、それをログファイル７０３（ｂ）に付加する。続いて、システムコールエンジン７１５は、ログファイル７０３（ｂ）内のメッセージの実行を再開する。

【００４６】ｌｉｂ．３ｄによって使用されるメッセージファンクションは、パイプ７０７のファイルディスクリプタを取得したのと同じようにパイプ７１０のファイルディスクリプタを取得する。次にメッセージファンクションは、パイプ７０７のファイルディスクリプタを使用してメッセージを送ることを試み、この試みが失敗すると、メッセージファンクションは再びパイププロセス７１１にパイプファイルディスクリプタを要求する。メッセージファンクションはパイプ７１０のファイルディスクリプタを受け取り、再びバックエンドに接続される。

【００４７】残りの故障シナリオは以下のように扱われる。

【００４８】・パイププロセス７１１が故障した場合。モニタ７１７が、故障を検出し、サーバを再起動する。新たに再起動されたプロセスは、ＷａｔｃｈＤによって保存されたプロセス状態からパイプ７１０への接続を取得する。他のプロセスはこの故障および回復について全く知らない。

【００４９】・システムコールエンジン７１５が故障し

た場合。
モニタ７１９が、故障を検出し、システムコールエンジン７１５を再起動する。ｌｉｂｆｔによって提供されるチェックポイントおよび回復のファンクションによって、新たに再起動されたシステムコールエンジン７１５は、外部ファイルから、前にチェックポイントしたステータスに回復することができる。他のプロセスはこの故障および回復について全く知らない。
【００５０】・バックエンドログプロセス７１６が故障した場合。
モニタ７１９が、故障を検出し、バックエンドログプロセス７１６を再起動する。今度も、プロセス７１６は、チェックポイントファイルからステータスを復元する。さらに、モニタ７１９は、モニタ７１７に、バックエンドログプロセス７１６が再起動されたことを通知し、続いてモニタ７１７は、パイププロセス７１１に通知する。次に、プロセス７１１は、パイプ７１０を、新しいバックエンドログプロセス７１６に接続する。各アプリケーションの次の書き込みは失敗し、ｌｉｂ．３ｄはパイププロセス７１１から新たな接続を取得する。
【００５１】・フロントエンドログプロセス７０５が故障した場合。
フロントエンドログプロセス７０５は、システム５１３の故障の期間中にのみ存在する。モニタ７１７は、フロントエンドログプロセス７０５の故障を検出すると、パイププロセス７１１に通知する。続いて、パイププロセス７１１は、フロントエンドログプロセス７０５を再起動し、それにパイプ７０８を再接続する。アプリケーションプログラム５０９の次の書き込みは失敗し、ｌｉｂ．３ｄ内のメッセージ送信ファンクションは、パイププロセス７１１から新たなパイプ７０８のファイルディスクリプタを取得する。
【００５２】［ユーザレベル名前空間４０５の実装：図８～図１１］ユーザレベル名前空間４０５は、カーネルサーバ３０５（ａ）によってアプリケーションプロセス５０３に提供されるファイルシステムからのファイルの任意のセットを指定するために使用することができる。図８に、カーネルサーバ３０５（ａ）によって提供されるファイルシステムの名前空間８０１と、ユーザレベルバックアップファイルシステム７０１内のユーザレベル名前空間４０５の間の関係を示す。
【００５３】名前空間８０１において、ファイル名はツリー（木）に配置される。図８のツリーの葉をなすファイル（Ｂ，Ｄ，Ｅ，Ｇ，Ｉ，Ｍ，Ｎ）はデータまたはプログラムを含む。残りのファイルは他のファイルのリストである。このようなファイルはディレクトリと呼ばれる。名前空間８０１内の任意のファイルは、カーネルサーバ３０５（ａ）に対して、パス名によって指定することが可能である。パス名は、ルート「／」で始まり、ルートからそのパス名によって指定されているファイルの名前までのすべてのファイルの名前を含む。従って、ファイルＤのパス名は／Ａ／Ｃ／Ｄであり、ファイルＬのパス名は／Ｊ／Ｋ／Ｌである。
【００５４】ユーザレベルバックアップファイルシステム７０１は、バックアップすべきファイルを、そのファイルを含む名前空間８０１のサブツリーを指定することによって指定する。次に、サブツリー内のファイルを変更するようなファイルに対する操作が、バックアップシステム５１３内のバックアップファイルに対して実行される。図８では、３つのサブツリー、８０３（ａ）、８０３（ｂ）、および８０３（ｃ）がバックアップすべきものとして選択されている。その結果、名前空間８０１内のデータファイルＤ、Ｅ、Ｇ、Ｉ、Ｍ、またはＮへの変更の結果として、そのデータファイルに対するバックアップファイルへの変更が行われ、ディレクトリＣ、Ｆ、Ｈ、およびＬへの変更も同様にそれらのバックアップファイルへの変更を引き起こす。サブツリー内のすべてのファイルがバックアップされるため、バックアップすべきファイルは、ユーザレベル名前空間４０５ではそのサブツリーのルートであるディレクトリのパス名によって指定することが可能である。こうして、サブツリー８０３（ａ）はユーザレベル名前空間４０５ではパス名／Ａ／Ｃ（８０５（ａ））によって指定される。
【００５５】もちろん、ユーザレベル名前空間４０５は、カーネルサーバ３０５（ｂ）によってシステムコールエンジン７１５に提供されるファイルシステムにもマップされなければならない。これはバックエンドマップ５１７によって行われる。図９に示したように、バックエンドマップ５１７は、ユーザレベル名前空間４０５における各オープンファイルに対するエントリ９０１を含む。このエントリは２つの部分を有する。ユーザレベル名前空間情報９０３は、ユーザレベル名前空間４０５におけるファイルを指定し、バックアップシステム情報９０５は、カーネルサーバ３０５（ｂ）によって提供されるファイルシステムにおいて、ユーザレベル名前空間情報によって指定されたファイルに対応するファイルを指定する。
【００５６】バックエンドマップ５１７により、カーネルサーバ３０５（ｂ）がバックエンドログプロセス７１６およびシステムコールエンジン７１５に提供するファイルシステムの名前空間９０７のサブツリーに、名前空間８０１のサブツリーをマップすることが可能となる。このマッピングは、名前空間８０１のサブツリーのルートのパス名を、名前空間９０７の対応するサブツリーのルートのパス名にマップすることによって行われる。ルートのパス名は、サブツリー内ではファイルのパス名のプレフィクスと呼ばれる。こうして、サブツリー８０３（ａ）におけるパス名はプレフィクス／Ａ／Ｃを有し、サブツリー８０３（ａ）内のファイルＥのパス名はＥとなる。名前空間９０７では、名前空間８０１のプレフィ

19

クス／A／Cを名前空間９０７のプレフィクス／Zにマップすることによって、サブツリー９０９はサブツリー８０３（a）に対応するようになる。マッピングをした後は、名前空間８０１においてパス名／A／C／Eによって指定されるファイルの変更の結果、名前空間９０７においてパス名／Z／Eによって指定されるファイルの変更が行われることになる。

【００５７】［フロントエンド複製ツリー５０５の詳細：図１０］好ましい実施の形態では、ユーザレベル名前空間４０５はフロントエンド複製ツリー５０５として実装される。図１０に、フロントエンド複製ツリー５０５の詳細を示す。フロントエンド複製ツリー５０５の２つの主要な要素は、ＲＴＲＥＥ１０１５およびファイルディスクリプタ（ＦＤ）キャッシュ１０２７である。ＲＴＲＥＥ１０１５は、バックアップすべきファイルからなるサブツリー８０３のルートのパス名の連結リストである。ファイルディスクリプタキャッシュ１０２７は、ファイルディスクリプタをデバイスおよびｉノード識別子に関係づける配列である。この実装の形式は、ＵＮＩＸオペレーティングシステムによって提供されるファイルシステムがファイルを３通りの方法で、すなわち、パス名によって、整数のファイルディスクリプタによって、および、ファイルが存在するデバイスの識別子とＵＮＩＸファイルシステムテーブル内のそのファイルに対するエントリ（ｉノード）とによって、指定することの結果である。ファイルのファイルディスクリプタは、そのファイルをオープンしたプロセスに対してのみ、かつ、そのプロセスがそのファイルをオープンしている間にのみ有効である。ＵＮＩＸファイルシステムテーブルでは、パス名とデバイスおよびｉノードとの間、ならびに、デバイスおよびｉノードと現在のファイルディスクリプタとの間の変換は可能であるが、パス名と現在のファイルディスクリプタとの間の直接の変換はできない。

【００５８】さらに詳細に説明すると、ＭＡＸＴＲＹ１００３およびＩＮＩＴ１００５は、フロントエンド複製ツリー５０５を初期化する際に使用される。ＭＡＸＴＲＹ１００３は、初期化ファンクションが、バックアップシステム５１３へのパイプ７１０の設定を試みてあきらめるまでの回数を示す。ＩＮＩＴ１００５は、パイプが設定されたかどうかを示す。ＲＰＬＯＰ配列１００９は、複製ツリー５０５に対して実行可能な操作の名前１０１１の配列である。

【００５９】ＲＴＲＥＥ　ＰＴＲ１０１３は、ＲＴＲＥＥリスト１０１５の第１要素へのポインタである。ＲＴＲＥＥリスト１０１５は、複製ツリー８０３ごとに１要素１０１７を含む連結リストである。各要素１０１７は、複製ツリー８０３のルートのパス名１０２１、パス名１０２１の長さ１０１９、およびこの連結リストにおける次の要素へのポインタ１０２３を含む。接続サーバ１０２５は、バックアップシステム５１３へのパイプ７１０の、名前空間８０１におけるパス名である。

20

【００６０】ＦＤキャッシュ１０２７は、ファイルディスクリプタキャッシュエントリ１０２９の配列である。この配列には、アプリケーションプロセス５０３に利用可能なファイルディスクリプタと同じ数だけ、エントリ１０２９がある。ＦＤキャッシュ１０２７内の与えられたファイルディスクリプタに対するエントリのインデックスはそのファイルディスクリプタである。エントリ１０２９は、そのエントリが現在有効であるかどうかを示し、かつ、そのファイルがオープンであった間に子プロセスを作成したかどうかをも示すステータスフラグを含む。また、エントリ１０２９は、主システム５１１においてそのファイルが存在するデバイスの識別子１１０１と、主システム５１１におけるそのファイルのｉノードの識別子１１０３とを含む。ＲＴＲＥＥ１０１５内のエントリによって指定されるサブツリー８０３には、現在オープンのファイルごとに有効なエントリ１０２９が存在する。

【００６１】［バックエンドマップ５１７の詳細］バックエンドマップ５１７は２つの部分、すなわち、パス名マップ１１１３およびオープン複製ファイルリスト１１１７を有する。パス名マップ１１１３は単に、主システム５１１の名前空間８０１内のパス名を、バックアップシステム５１３の名前空間９０７内のパス名にマップする。マップ内の各エントリ１１１５は、フロントエンドパス名１１１８とバックエンドパス名１１１９の間の関係を確立する。パス名マップ１１１３には、フロントエンド名前空間９０７内のサブツリー８０３のルートを、名前空間９０７内のサブツリーのルートにマップするエントリが含まれる。バックエンドパス名１１１９はバックエンドシステム情報９０５の一部である。好ましい実施の形態では、これらのマッピングは、システム設定ファイルで指定される。

【００６２】オープン複製ファイルリスト１１１７は、アプリケーションプロセス５０３が複製ツリー８０３において現在オープンしている各ファイルに対するエントリ１１２０を含む。エントリ１１２０内のユーザレベル名前空間情報９０３は、フロントエンドファイル識別子（ＦＦＩＤ）１１０５およびフロントエンドパス名（ＦＰ）１１０６を含む。フロントエンドファイル識別子１１０５は、主システム５１１内のファイルに対するデバイス識別子およびｉノード識別子からなる。フロントエンドパス名１１０６は、フロントエンドプレフィクス（ＦＰＲ）１１０７およびサブツリーパス名１１０８に分けられる。フロントエンドプレフィクス（ＦＰＲ）１１０７は、フロントエンド名前空間８０１における当該ファイルのサブツリーに対するプレフィクスである。サブツリーパス名１１０８は、サブツリーにおけるファイルのパス名である。エントリ１１２０内のバックアップシステム情報９０５は、バックエンドファイルディスク

リプタ１１１１からなる。バックエンドファイルディスクリプタ１１１１は、カーネルサーバ３０５（ｂ）によって提供されるファイルシステムにおける当該ファイルのファイルディスクリプタである。好ましい実施の形態では、バックエンドマップ５１７は、フロントエンドファイル識別子１１０５およびフロントエンドパス名１１０６のいずれによってもアクセス可能なハッシュテーブルとして実装される。

【００６３】［データ構造体５０５および５１７に関する操作］以下では、どのようにしてデータ構造体５０５および５１７を作成するか、および、これらのデータ構造体がさまざまなファイル操作によってどのように影響を受けるかを説明する。好ましい実施の形態では、アプリケーションプロセス５０３は、Ｋｏｒｎシェルを使用するＵＮＩＸオペレーティングシステム上で時刻される。Ｋｏｒｎシェルによれば、プロセスは、当該プロセスがＫｏｒｎシェルを呼び出すときにはいつも実行されるファイルを指定するＥＮＶ変数を設定することが可能である。アプリケーションプロセス５０３においてＥＮＶ変数によって指定されるファイルは、アプリケーションプロセス５０３が、フロントエンド複製テーブル５０５を構成し初期化するのに必要な情報を含む。いったん作成されると、テーブル５０５は、アプリケーションプロセス５０３のアドレス空間の一部となり、ＵＮＩＸオペレーティングシステムのｆｏｒｋシステムコールで作成されそれによって親の環境を継承する、アプリケーション５０３の任意の子プロセスに利用可能となる。他方、ｅｘｅｃシステムコールは、子プロセスに新しい環境を与える。ｅｘｅｃシステムコールで作成されるアプリケーションプロセス５０３の子プロセスにフロントエンド複製ツリー５０５が利用できるようにするため、ｌｉｂ．３ｄは、フロントエンド複製ツリー５０５を新しいプロセスのＥＮＶ変数にコピーするｅｘｅｃファンクションを有する。これにより、その新しいプロセスは、親のアドレス空間を継承していなくても、フロントエンド複製ツリー５０５を利用することができる。他の実施の形態では、ｅｘｅｃによって作成される子プロセスにフロントエンド複製ツリー５０５を渡すために、名前付きパイプまたは外部ファイルを使用することも可能である。

【００６４】ファイル操作の説明に進むと、第１のファイル操作はｍｏｕｎｔ（マウント）操作である。ＵＮＩＸオペレーティングシステムでは、ｍｏｕｎｔはファイルシステムからの名前のツリーを、オペレーティングシステムの名前空間に追加する。好ましい実施の形態では、ｌｉｂ．３ｄで実装されるｍｏｕｎｔのバージョンは、フロントエンド名前空間８０１のサブツリーが複製ツリー８０５としてユーザレベル名前空間４０５に追加されるモードを有する。ｍｏｕｎｔがこのモードで使用されるとき、パス名引数は、ユーザレベル名前空間４０５に追加されるサブツリー８０３のルートのパス名である。ファンクションは、そのパス名に対する複製ツリーエントリ１０１７を作成し、そのエントリを複製ツリー１０１５に追加することによって、サブツリー８０３をユーザレベル名前空間４０５に追加する。また、指定されたパス名を有する複製ツリーエントリ１０１７を複製ツリー１０１５から削除するｕｍｏｕｎｔ（マウント解除）操作もある。

【００６５】アプリケーションプロセス５０３が複製ツリー８０５内のファイルに対してオープン操作を実行すると、ｌｉｂ．３ｄ内のオープンファンクションは、新たにオープンされるファイルに対するファイルディスクリプタキャッシュエントリ１０２９を作成し、オープンメッセージをバックエンドログプロセス７１６へ送る。このオープンメッセージは、オープンしたファイルの主システム５１１におけるパス名、デバイス識別子、およびｉノード識別子を含む。このメッセージがシステムコールエンジン７１５によって実行されると、その結果、バックエンドマップ５１７内にエントリ９０１が作成される。パス名マップ１１１３を使用して、主システム５１１内のオープンされているファイルに対応するバックエンドシステム５１３内のファイルが発見され、対応するファイルに対するファイルディスクリプタがバックエンドファイルディスクリプタ１１１１に入れられる。

【００６６】ファイルがオープンされると、主システム５１１におけるファイル操作は、そのファイルを識別するファイルディスクリプタを使用する。バックアップシステム５１３内のバックアップファイルに対する対応する操作に対するメッセージは、デバイス識別子およびｉノード識別子を使用してファイルを識別する。このようなメッセージを実行するためには、システムコールエンジン７１５は、メッセージで指定されるデバイスおよびｉノードに対するオープン複製ファイルリスト１１１７内のエントリ１１１９にアクセスするだけでよい。このエントリは、バックアップシステム５１３における操作を実行するのに必要なファイルディスクリプタ１１１１を含む。

【００６７】アプリケーションプロセス５０３が複製ツリー５０５内のファイルをクローズすると、ｌｉｂ．３ｄのクローズファンクションは、ステータスフィールド１０３３から、子プロセスがそのファイルを使用しているかどうかを判断する。どの子プロセスも使用していない場合、クローズファンクションは、複製ツリー５０５内のそのファイルに対するファイルディスクリプタキャッシュエントリ１０２９を無効にし、デバイス識別子およびｉノード識別子を含むクローズメッセージをバックアップシステム５１３へ送る。システムコールエンジン７１５は、このメッセージを実行するとき、デバイス識別子およびｉノード識別子を使用してこのファイルに対するエントリ１１１９を見つける。続いて、このファイ

ルを識別するためにバックエンドファイルディスクリプタ１１１１を使用して、バックアップシステム５１３内のファイルをクローズし、最後に、オープン複製ファイルリスト１１１７からエントリ１１１９を削除する。

【００６８】［ユーザレベルバックアップファイルシステムを使用した複製ファイルの実装：図１３］バックアップファイルシステム５０１は、アプリケーションプロセス５０３のフロントエンド複製ツリー５０５において指定される、主システム５１１からの各ファイルの現在のコピーが、バックアップシステム５１３上に存在することが保証されるという点で有効である。しかし、主システム５１１からのファイルのコピーを変更するバックアップシステム５１３における操作の結果は、主システム５１１内のファイルには反映されない。実際に、このことは、主システム５１１しか、バックアップシステム５１３上にバックアップされているファイルを変更することができないことを意味する。

【００６９】ファイルが複製ファイルである場合に必要とされるように、主システム５１１およびバックアップシステム５１３がいずれもファイルのコピーを変更することができるためには、各システムは、互いのシステム上でなされる変更をバックアップしなければならない。すなわち、２つのシステムは、複製ファイルのコピーに対する操作に関してピアでなければならない。図５に関していえば、２つのシステム５１１および５１３はそれぞれ、他方のシステムへのバックアップメッセージのためのチャネル５１２と、バックエンドサーバ５１５とを有していなければならない。さらに、ファイルを変更したいシステム上のプロセスは、ｌｉｂ．３ｄ（５０７）と、ファイルが複製ファイルとしてリストされたフロントエンド複製ツリー５０５を有していなければならない。さらに、複製ファイルのコピーの変更が両方のシステムにおいて同じ順序で起こること、および、複製ファイルのローカルコピーに対する読み出し操作が、複製ファイルのリモートコピーでなされた書き込みを考慮に入れて提供されることを確実にするため、同期システムが要求される。

【００７０】図１３に、２つのピアホスト１３０２（Ａ）および１３０２（Ｂ）ならびに複製ファイル１３２５を有する分散システム１３０１の概観を示す。各ホストはカーネルサーバ３０５（図示せず）および大容量記憶装置を有する。大容量記憶装置は、ここでは、ホスト１３０２（Ａ）に対してはディスク３０７（ａ）であり、ホスト１３０２（Ｂ）に対してはディスク３０７（ｂ）である。各ディスクは複製ファイル１３２５の同一のコピーを有する。ホスト１３０２（Ａ）上のコピーはコピー１３２５（Ａ）であり、ホスト１３０２（Ｂ）上のコピーはコピー１３２５（Ｂ）である。さらに、各ホスト１３０２は、バックエンドサーバ５１５を有し、他方のホスト１３０２からバックアップメッセージを受け取ることができる。ホスト１３０２（Ａ）における３つのプロセス１３０９（Ａ，１..３）はｌｉｂ．３ｄコード５０７を含み、ファイル１３２５を複製ファイルとして指定するフロントエンド複製ツリー５０５を有する。ホスト１３０２（Ｂ）上の１つのプロセス１３０９（Ｂ，１）はコード５０６およびそのようなフロントエンド複製ツリーを有する。各ホスト１３０２は互いにバックアップとして機能するため、プロセス１３０９（Ａ，１..３）が書き込み操作（すなわち、ホスト１３０２（Ａ）上の複製ファイル１３２５のコピー１３２５（Ａ）を変更する操作）を実行するごとに、その書き込み操作の結果、バックアップメッセージ５１２（Ａ）が生じ、ホスト１３０２（Ｂ）上のバックエンドサーバ５１５（Ｂ）はこれに応答して、複製ファイルのコピー１３２５（Ｂ）に対して同じ書き込み操作を実行する。プロセス１３０９（Ｂ，１）がコピー１３２５（Ｂ）に対して書き込み操作を実行すると、この書き込み操作の結果バックアップメッセージ５１２（Ｂ）が生じ、バックエンドサーバ５１５（Ａ）はこれに応答して、コピー１３２５（Ａ）に対して同じ書き込み操作を実行する。バックアップメッセージ５１２は、送信された順にメッセージが到着することを保証するチャネルを通じて送られ、その結果、コピー１３２５（Ａ）および１３２５（Ｂ）に対する書き込み操作は同じ順序で行われる。このようなチャネルを実装する１つの方法は、ＴＣＰ／ＩＰを通じてバックアップメッセージ５１２を送ることである。

【００７１】もちろん、バックエンドサーバ５１３は、プロセス１３０９が複製ファイル１３２５（Ａ）に対して書き込み操作を実行するのと同時に複製ファイル１３２５（Ｂ）に対して書き込み操作を実行するわけではない。その結果、ファイル１３２５（Ｂ）に対する読み出し操作は、ファイル１３２５（Ａ）に対する同時の読み出し操作と異なる結果となる可能性がある。ある場合には、これは異ならないが、他の場合には異なることがある。その結果、システム１３０１には、複製ファイル１３２５に対する２種類の読み出し操作が可能である。第１の読み出し操作は「アトミック読み出し」操作である。アトミック読み出し操作は、複製ファイル１３２５のコピーに一貫性がある必要がないときに使用される。この操作は単に、複製ファイル１３２５のローカルコピーに対する現在の書き込み操作が終了するまで待機してからそのローカルコピーを読み出す。第２の読み出し操作は「順次読み出し」操作である。この操作は、複製ファイル１３２５のコピーに一貫性がなければならない場合に使用され、従って、読み出されている複製ファイルのコピーは、その複製ファイルの他のすべてのコピーと一貫性があるように、複製ファイルに対する書き込み操作と同期している。

【００７２】書き込み操作と順次読み出し操作の同期

25

は、複製ファイル１３２５に対する２つのトークン、すなわち、書き込みトークン１３２７および読み出しトークン１３２８によって実現される。書き込みトークン１３２７を有するホスト１３０２は、複製ファイルのローカルコピーに対する読み出し操作または書き込み操作を実行することが可能である。読み出しトークン１３２８を有するホスト１３０２は、ローカルコピーに対する読み出し操作を実行することは可能であるが、書き込み操作を実行することはできない。いずれのトークンも有しないホスト１３０２は、アトミック読み出し操作のみ実行可能である。ホスト１３０２は、必要なトークンを有しない場合、他のホスト１３０２にそのトークンを要求する。他のいずれかのホスト１３０２で書き込み操作が未完了である場合、最後の書き込みバックアップメッセージ５１２を送った後に、バックアップメッセージ５１２のために使用したチャネルにトークンを送る。このようにトークンを送ることによって、ホスト１３０２は、複製ファイル１３２５のすべてのローカルコピーにおいて書き込み操作が同じ順序で起こること、および、複製ファイル１３２５の同一のローカルコピーに対して順次読み出しが実行されることを保証する。

【００７３】与えられた瞬間にはただ１つのホスト１３０２のみが書き込みトークン１３２７を有し、その瞬間には、他のすべてのホスト１３０２はトークンを有しない。いずれのホストも書き込みトークン１３２７を有しない場合、すべてのホストは読み出しトークン１３２８を有する。読み出しトークンおよび書き込みトークンのいずれも有しないホスト１３０２はいずれかを要求することが可能である。書き込みトークンを有するホストは、読み出しトークンまたは書き込みトークンのいずれかを与えることが可能である。読み出しトークンを有するホストは書き込みトークンを要求または授与することが可能である。

【００７４】システム１３０１が３つ以上のホスト１３０２を有するとき、書き込み操作は、書き込みトークン１３２７を有しないすべてのホスト１３０２へ同報される。トークンの要求およびトークンの授与もまたすべてのホスト１３０２に同報される。同報は、要求および授与の信頼性のある同報順序を提供する信頼性のある同報パッケージを使用して行われる。このようなパッケージの一例は、コーネル大学によって提供されているＩＳＩＳである。ＩＳＩＳは、「故障がある場合の信頼性のある通信(Reliable Communication in the Presence of Failures)」、ACM Transactions on Computer Systems, 5, 1、１９８７年２月、第４７～７６ページ、に記載されている。読み出しトークン１３２８の場合、読み出しトークンを授与することができる唯一のホスト１３０２は、書き込みトークンを有するホストである。その結果、読み出しトークンを授与するメッセージが複数存在することはない。書き込みトークン１３２７の場合、書き込みトークン１３２７を有する単一のホスト１３０２が存在するか、または、すべてのホストが読み出しトークン１３２８を有する。前者の場合、書き込みトークン１３２７を授与するメッセージはただ１つ存在する。後者の場合、要求中のホスト１３０２は、書き込みトークン１３２７を実際に有する前に、読み出しトークン１３２８を有するすべてのホスト１３０２から授与メッセージを受け取らなければならない。

26

【００７５】プロセス１３０９が自己のホスト１３０２上の複製ファイル１３２５のコピーに書き込みをするためには、２つの条件が満たされなければならない。

・このプロセスが実行されているホスト１３０２は、複製ファイルに対する書き込みトークン１３２７を有していなければならない。

・ホスト１３０２内の複製ファイル１３２５のコピーに対する他のホスト１３０２からの未完了の書き込み操作があってはならない。

【００７６】ホスト１３０２は、書き込みトークン１３２７を有しない場合、他のホストに書き込みトークン１３２７を要求しなければならない。他のホスト１３０２はバックアップメッセージ５１２において書き込みトークン１３２７を送り、それによって、第２の条件が満たされること、すなわち、受信側ホスト１３０２は最後のバックアップメッセージ５１２で指定される変更が完了するまで複製ファイル１３２５のコピーを変更しないことを保証する。

【００７７】好ましい実施の形態では、書き込みトークン１３２７を使用した同期は、各ホスト１３０２上のトークンサーバ１３１１と、トークンファイル１３０７と、バックアップメッセージ５１２を受信した順序で送出するチャネルとによって実現される。トークンファイル１３０７は、ホスト１３０２上にコピーを有する各複製ファイル１３２５に対する領域を有する。標準的なオペレーティングシステムのロッキングサブシステムでは、ファイルの領域をロックすることが可能である。２種類のロックがある。排他ロックでは、ただ１つのプロセスのみがそのファイルにアクセスすることが可能である。共有ロックでは、任意数のプロセスがアクセスすることができる。一般に、プロセスは、領域に書き込むためにはその領域に排他ロックを有し、領域から読み出すためには共有ロックを有していなければならない。好ましい実施の形態では、トークンファイル１３０７における複製ファイル１３２５の領域に対するオペレーティングシステムロックを使用して、その複製ファイル１３２５に対する書き込み操作と順次読み出し操作を同期させるために使用されるトークンを実現する。

【００７８】例えば、好ましい実施の形態では、書き込みトークンは、複製ファイル１３２５の領域に対するオペレーティングシステムロックから形成される書き込みトークンロックとして実現される。トークンサーバ１３

１１は、書き込みトークンを有することを示すメッセージを受け取ると、書き込みトークンロックを獲得する。トークンサーバ１３１１が書き込みトークンロックを有する限り、ホスト１３０２上で実行中のプロセス１３０９は、複製ファイルのローカルコピーに対するアトミック読み出し操作、順次読み出し操作、または書き込み操作のロックを獲得することができる。これらのロックもまた、トークンファイル１３０７におけるオペレーティングシステムロックを使用して実現される。

【００７９】他のホスト１３０２が書き込みトークンを要求すると、トークンサーバ１３１１は書き込みトークンロックを解放し、トークンなしロック（他のロックと同様に実現される）を獲得する。トークンサーバ１３１１がトークンなしロックを有する限り、ホストシステム１３０２において複製ファイル１３２５に書き込むことが可能な唯一のプロセスはバックエンドサーバ５１５である。もちろん、バックエンドサーバ５１５は、現在書き込みトークン１３２７を有するホスト１３０２からのバックアップメッセージ５１２に応答する。

【００８０】システム１３０１の動作は以下の通りである。ユーザレベルバックアップファイルシステム５０１の説明で既に述べたように、ｌｉｂ．３ｄ（５０７）は、複製ファイルに対する操作を実行するアプリケーションプロセス５０９のコードに静的にまたは動的にバインドされる。その後、ファイルがフロントエンド複製ツリー５０５において複製ファイルとして指定される。システム１３０１で使用されるｌｉｂ．３ｄ（５０７）のバージョンは、標準的なＩ／Ｏライブラリ書き込みルーチンを、図１２に示す書き込み操作で置き換える。第３行のget_write_token()関数１２０１は、関連するホストのトークンサーバ１３１１に書き込みトークン１３２７を要求する。そのホストのトークンサーバ１３１１が書き込みトークン１３２７を有する場合、この関数は直ちに復帰する。トークンサーバ１３１１は、書き込みトークン１３２７を有していない場合、他のホストに要求し、書き込みトークンが到着すると復帰する。トークンサーバ１３１１が書き込みトークン１３２７を有すると、プロセス１３０９は第４行で書き込みシステムコールsyscall(SYS_write, fildes, buf, nbyte)を実行する。その後、システム５０１の説明で述べたように、関数は、フロントエンド複製ツリー５０５から、ファイルが複製されているかどうかを判断する。複製されている場合、書き込みメッセージ５１２が他のホスト１３０２に送られ、書き込みトークンは解放される（１２０３）。書き込みトークン１３２７は、同じようにして、複製ファイル１３２５を変更するいずれのホスト１３０２上の書き込み操作に対しても、獲得されなければならない。その結果、すべての変更は複製ファイル１３２５のすべてのコピーに対して行われ、すべての変更は同じ順序で行われる。

【００８１】［同期の詳細な実装：図１４］好ましい実施の形態では、与えられたホスト１３０２上の複製ファイル１３２５のコピーに属するロックファイル１３０７の領域は、書き込み操作に関連する２つのロックを有する。第１のロックは、書き込みトークン１３２７がホスト１３０２上にあるかどうかを示し、第２のロックは、複製ファイル１３２５のコピーが、当該ホスト１３０２上のプロセス１３０９による書き込みに利用可能であるかどうかを示す。図１４に、好ましい実施の形態においてこれら２つのロックをどのようにして使用するかを示す。図の擬似コード１４０１は今度もｌｉｂ．３ｄ（５０７）の書き込み操作に対するものである。ロックを含む領域は変数TOKEN_REGION（１４０３）によって表され、これは２つのフィールドを有する。ロックのSTATEは、書き込みトークン１３２７がホスト１３０２上にあるかどうかを示し、ロックのTOKENは、プロセス１３０９が書き込みを実行することができるかどうかを示す。STATEによって表されるロックは、トークン１３２７が他のホスト１３０２上にあるとき、ローカルトークンサーバ１３１１によって排他ロックされたまま保持される。

【００８２】擬似コード１４０１によって記述される動作は以下の通りである。第３行に示されるように、ロック１４０３を含むトークンファイル１３０７の領域が関数fd2tokenによって検索される。この関数は、複製ファイル１３２５のローカルコピーのファイルディスクリプタをとり、領域１４０３を返す。次のステップで、複製ファイル１３２５に対する書き込みトークン１３２７がローカルホスト１３０２内にあるかどうかを判定する。これは、第４行で、領域１４０３のSTATEフィールドの非ブロッキング共有ロックを要求することによって行われる。このロックが取得可能である場合、書き込みトークン１３２７はローカルホスト１３０２上にある。このロックが取得可能でない場合、擬似コード１４０１は、トークンサーバ１３１１が他のホスト１３０２上の対応するトークンサーバにトークン１３２７を要求するメッセージを送り、そのトークンを提供するメッセージが返るのを待機するようにする関数（図示せず）を呼び出す。第４行では、書き込みトークン１３２７がローカルホスト１３０２上にあるかどうかを複数のプロセス１３０９が判定できるように、共有ロックの取得を試みる。

【００８３】書き込みトークン１３２７がローカルに利用可能となると、次のステップへ進む。第６行に示されるように、領域１４０３のSTATEフィールドに対してもう１つのロック要求がなされる。今度はこれはブロッキングであり、コード１４０１を実行しているプロセス１３０９は、STATEに対する共有ロックを取得することができるまで（すなわち、書き込みトークン１３２７がローカルに利用可能になるまで）待機し（第６行）、その後、TOKENに対する排他ロックを獲得するまでブロック

する。プロセス１３０９が排他ロックを受け取ると、実際に書き込みシステムコールがなされ、メッセージ５１２が、複製ファイル１３２５のローカルコピーへの書き込みの内容とともに、他方のホスト１３０２へ送られる。これが行われると、領域１４０３はロック解除され、書き込み操作が終了する。

【００８４】もちろん、ローカルホスト１３０２上で複製ファイル１３２５に対する書き込みを試みている他のいずれのプロセス１３０９も、STATEに対する共有ロックを有し、TOKENに対する排他ロックを待機しているプロセスの待ち行列に入ることができる。書き込みを完了したプロセス１３０９がTOKEN_REGIONをロック解除すると、待ち行列における次のこのようなプロセスがTOKENに対する排他ロックを取得し書き込み操作を実行することができる。さらに、ローカルトークンサーバ１３１１が、他のトークンサーバ１３１１から複製ファイル１３２５に対する書き込みトークン１３２７を要求するメッセージ５１２を受け取ると、ローカルトークンサーバ１３１１はSTATEに対する排他ロックを要求する。ローカルトークンサーバ１３１１は、STATEに対する共有ロックを有するすべてのプロセス１３０９が書き込み操作を完了した後にのみ、その排他ロックを受け取る。ローカルトークンサーバは、STATEに対する排他ロックを受け取ると、書き込み操作によって生成されたメッセージが他のホスト１３０２に送られたのと同じチャネルによって、そのことを知らせるメッセージを他のホスト１３０２へ送る。チャネルに入れられたメッセージは、送られた順序で到着し、その結果、他のホスト１３０２上のトークンサーバ１３１１は、当該他のホスト１３０２上のバックエンドサーバ５１５が、トークン１３２５を有していたホスト１３０２からのすべての書き込みメッセージ５１２を処理した後にのみ、STATEに対する排他ロックを解放する。

【００８５】他のホスト１３０２上で実行される順次読み出し操作と書き込み操作の同期は、書き込みトークン１３２７に関して説明したのとほぼ同様に、読み出しトークン１３２８によって達成される。順次読み出しを実行するプロセス１３０９は、読み出しトークン１３２８または書き込みトークン１３２７がホスト１３０２上にあるかどうかを示すトークンファイル１３０７の一部に対する共有ロックを取得することをまず試みる、１ｉｂ．３ｄ（５０７）内のコードを実行する。この試みが失敗した場合、プロセス１３０９は、トークンサーバ１３１１が読み出しトークン１３２８を他のホスト１３０２から取得することを要求する。トークンは、書き込みトークン１３２７について説明したのと同様にして取得される。次に、プロセス１３０９は、そのトークンを表す領域に対する排他ロックを取得することを試み、読み出しトークン１３２８がホスト１３０２に到着するまでブロッキングする。トークンが到着すると、プロセス１３０９は、複製ファイル１３２５のローカルコピーに対する共有ロックを要求する。プロセス１３０９は、そのローカルコピーが他のプロセス１３０９によって実行されているローカル書き込み操作に対する排他ロックでなく、かつ、バックアップメッセージ５１２に応答してバックアップサーバ５１５によって実行されているリモート書き込み操作に対する排他ロックでもない場合にのみ、その共有ロックを受け取ることができる。

【００８６】既に示したように、書き込みトークン１３２７を有するトークンサーバ１３１１は読み出しトークン１３２８を授与することができる。書き込みトークン１３２７を有するトークンサーバ１３１１が要求を受け取った場合、複製ファイルに対する書き込み操作が終了するのを待ち、複製ファイルのローカルコピーに対するロックを排他ロックから共有ロックに変更し、書き込みバックアップメッセージ５１２のために使用しているチャネルによって読み出しトークンを送る。このことすべてにより、読み出しトークン１３２８は、最後の書き込みバックアップメッセージ５１２の後に、要求側ホスト１３０２に到着する。

【００８７】［複製ファイルに対する状態マシン：図１５］ホスト１３０２内のアプリケーションプロセス１３０９、トークンサーバ１３１１、およびバックエンドサーバ５１５の協力ならびにホスト１３０２間でのトークンサーバ１３１１の協力は、プロセス１３０９、トークンサーバ１３１１、およびバックエンドサーバ５１５を状態マシンとして考察することによってより良く理解される。与えられた複製ファイル１３２５に関して、これらの各マシンの状態は、そのファイルに対する書き込みトークン１３２７および読み出しトークン１３２８に依存し、トークンがホスト１３０２間を移動するにつれて変化する。

【００８８】アプリケーションプロセス１３０９は、複製ファイル１３２５に関して４つの状態を有する。

【００８９】１．操作なし状態。この状態では、アプリケーションプロセス１３０９は、複製ファイル１３２５のローカルコピーに対するいかなる種類のロックも有しておらず、従って、ローカルコピーに対する読み出し操作も書き込み操作を実行することができない。

【００９０】２．アトミック読み出し状態。この状態では、プロセス１３０９はローカルコピーに対する共有ロックのみを有し、従って、アトミック読み出し操作のみを実行することができる。

【００９１】３．順次読み出し状態。この状態では、複製ファイル１３２５に対する読み出しトークン１３２８または書き込みトークン１３２７がホスト１３０２にあり、プロセス１３０９はローカルコピーに対する共有ロックを有し、従って、アトミック読み出しのみならず順次読み出し操作を実行することができる。

【００９２】４．書き込み状態。この状態では、複製フ

31

ァイル１３２５に対する書き込みトークン１３２７がホスト１３０２にあり、プロセス１３０９はローカルコピーに対する排他ロックを有し、従って、順次読み出しおよびアトミック読み出し操作のみならず書き込み操作を実行することができる。書き込み操作は、複製ファイル１３２５の他のコピーでバックアップされる。

【００９３】状態の説明から明らかなように、ある状態から他の状態への遷移には、トークンおよびロックの獲得および喪失が伴う。例えば、操作なし状態から順次読み出し状態への遷移は、ホスト１３０２における読み出しトークン１３２８の獲得およびプロセス１３０９による共有ロックの獲得を要求する。

【００９４】バックエンドサーバ５１５を実現するプロセスの状態は、アプリケーションプロセス１３０９の状態１および４と密接に関係している。

【００９５】１．操作なし状態。この状態では、バックエンドサーバ５１５は、複製ファイル１３２５のローカルコピーに対するいかなる種類のロックも有しておらず、従って、ローカルコピーに対する読み出し操作も書き込み操作を実行することができない。

【００９６】２．書き込み状態。この状態では、バックエンドサーバ５１５は、複製ファイル１３２５のローカルコピーに対する排他ロックを有し、従って、ローカルコピーに書き込みをすることができる。

【００９７】以上のことからわかるように、状態変化は、ホスト１３０２間のトークンの移動によって引き起こされる。

【００９８】最も複雑な場合は、トークンサーバ１３１１（Ａ）および（Ｂ）を実現するプロセスの場合である。トークンサーバ１３１１は、ホスト１３０２（Ａ）と（Ｂ）の間でトークンを渡すために相互に協力しなければならない。図１５は、２つのトークンサーバ１３１１に対する状態図である。図１５において、各状態は番号１５０１、１５０２、...、１５０６を有する。状態遷移は矢印で示される。矢印の参照番号の最後の２桁は、その矢印で示される遷移がなされる始状態および終状態を示す。従って、矢印１５３１は、状態１５０３から状態１５０１への遷移を示す。各状態遷移は、トークンサーバ１３１１で受け取られるメッセージの結果であり、これにより、メッセージが他のトークンサーバ１３１１に送られることもある。

【００９９】図１５において、与えられた遷移に対して受け取られるメッセージは、その遷移の矢印のそばに斜体字で示され、送られるメッセージはブロック体で示される。例えば、矢印１５３１で示される遷移は、他のトークンサーバ１３１１からのTSgetRtokenTS（斜体）メッセージの結果である。また、この遷移は、他のトークンサーバへのTSgrantRtokenRP_TS（ブロック体）メッセージを生成する。メッセージの名前は、メッセージの始点、宛先、内容、およびそのメッセージがとる経路を示す。例えば、TSgrantRtokenRP_TS（ブロック体）は、バックアップメッセージ５１２のために使用されるチャネルを通じて送られなければならない（RP_）、一方のトークンサーバ（第１のTS）から他方のトークンサーバ（第２のTS）への読み出しトークン授与（grantRtoken）メッセージである。同様に、TSgetRtokenTS（斜体）は、一方のトークンサーバから他方のトークンサーバへの読み出しトークン要求メッセージであるが、このメッセージは、バックアップメッセージ用のチャネルを通じて送られる必要はない。

32

【０１００】図１５の概観からはじめると、まず、３つの主要な状態がある。

【０１０１】・書き込みトークン状態１５０３。この状態では、ローカルホスト１３０２は書き込みトークン１３２７を有し、プロセス１３０９は複製ファイル１３２５のローカルコピーに対するすべての読み出し操作および書き込み操作を実行することが可能であり、トークンサーバ１３１１は読み出しトークンおよび書き込みトークンの両方を授与することが可能である。

【０１０２】・読み出しトークン状態１５０１。この状態では、ローカルホスト１３０２は読み出しトークンのみを有し、プロセス１３０９は複製ファイル１３２５のローカルコピーに対するすべての読み出し操作を実行することができるがローカルコピーへの書き込みはできず、トークンサーバ１３１１は書き込みトークン１３２７の要求または授与をすることができる。

【０１０３】・トークンなし状態１５０５。この状態では、ローカルホスト１３０２はトークンを有さず、プロセス１３０９は複製ファイル１３２５のローカルコピーに対するアトミック読み出し操作のみを実行することが可能であり、バックエンドサーバ５１５のみが複製ファイル１３２５のローカルコピーに書き込むことが可能であり、トークンサーバ１３１１は、読み出しトークンまたは書き込みトークンの要求のみをすることが可能である。

【０１０４】図１５における始状態は読み出しトークン状態１５０１である。アプリケーションプロセス１３０９が読み出しトークン状態１５０１の間に書き込み操作を試みると、状態遷移１５１２が起こる。トークンサーバ１３１１はプロセス１３０９からAPgetWtokenTS（斜体）要求を受け取り、この要求に応答して、他のホスト１３０２のトークンサーバ１３１１へTSgetWtokenTS（ブロック体）メッセージを送る。ここで、ローカルトークンサーバ１３１１は、読み出しトークン・書き込みトークン待機（RwaitW）状態１５０２において、他のトークンサーバから書き込みトークンを授与するTSgrantWtokenTS（斜体）メッセージを受け取るまで待機する。状態の名前が示しているように、ローカルホストは、書き込みトークンを待っている間、読み出しトークンを保持する。TSgrantWtokenTS（斜体）メッセージを受け取

33

ると、遷移１５２３が起こり、トークンサーバ１３１１は書き込みトークン状態１５０３に入り、ローカルホスト１３０２に書き込みトークン１３２７が来る。ここで、書き込みトークンに対する要求を生じた書き込み操作が実行され、その結果、複製ファイル１３２５のローカルコピーに書き込みが行われ、他のホスト１３０２へ書き込みバックアップメッセージ５１２が送られる。

【０１０５】もちろん、いずれのトークンサーバ１３１１も書き込みトークン１３２７を有さず、それぞれが他方から書き込みトークン１３２７を要求する可能性もある。その場合、状態１５０２においてタイブレークアルゴリズム（状態遷移１５２２によって表されている）が実行され、いずれのホスト１３０２が書き込みトークン１３２７を受け取るかが決定される。このアルゴリズムは、一方のホストが一次ホストとして指定され、他方のホストが二次ホストとして指定されることを要求する。一次ホストは、他方のホスト１３０２からの要求に応答して、その要求を無視して状態１５０３を続行する。二次ホストはトークンなし・書き込みトークン待機（NwaitW）状態１５０４への遷移１５２４を行う。

【０１０６】トークンサーバ１３１１は、状態１５０３にあるとき、書き込みトークン１３２７を有し、要求中のトークンサーバ１３１１へ書き込みトークン１３２７または読み出しトークン１３２８のいずれかを提供することができる。読み出しトークン１３２８に対する要求に応答して、矢印１５３１で示される遷移が起こる。TSgetRtokenTS（斜体）メッセージに応答して、トークンサーバ１３１１は書き込みトークン１３２７を放棄し、要求中のトークンサーバ１３１１へTSgrantRtokenRP_TS（ブロック体）メッセージを送るが、読み出しトークン１３２８は保持する。その結果、いずれのトークンサーバ１３１１も読み出しトークンを有し、状態１５０１にあることになる。

【０１０７】要求が、書き込みトークン１３２７に対するものであるとき、矢印１５３５で示される遷移が起こる。TSgetWtokenTS（斜体）要求に応答して、トークンサーバ１３１１は自己のトークンを放棄し、メッセージ５１２のために使用されるチャネルを通じてTSgrantWtokenRP_TS（ブロック体）メッセージをバックエンドサーバへ送り、トークンサーバ１３１１は状態１５０５に入る。状態１５０５は、状態１５０１から遷移１５１５によって到達することもある。この遷移は、状態１５０１にあるトークンサーバ１３１１がTSgetWtokenTS（斜体）メッセージを受け取り、それに応答して今遷移１５３５について説明したように動作するときに起こる。ただし、トークンサーバ１３１１のホストは書き込みをしているのではないので、メッセージ５１２のチャネルを通じてバックエンドサーバへメッセージを送る必要はない。

【０１０８】ローカルホスト１３０２上のアプリケーション

34

ョンプロセス１３０９が複製ファイル１３２５のローカルコピーに対する読み出し操作または書き込み操作を実行しようと試みるまで、トークンサーバ１３１１は状態１５０５にとどまる。読み出し操作の場合、この試みの結果、遷移１５５６が起こり、プロセス１３０９からのAPgetRtokenTS（斜体）メッセージがトークンサーバ１３１１によって受け取られ、トークンサーバ１３１１はこれに応答して他方のトークンサーバ１３１１へTSgetRtokenTS（ブロック体）メッセージを送る。次に、トークンサーバ１３１１は、トークンなし・読み出しトークン待機（NwaitR）状態１５０６に入り、読み出しトークン１３２８を待機する。待機中、書き込みトークン１３２７に対するローカル要求は待ち行列に入れる。このトークンを授与するTSgrantRtokenRP_TS（斜体）メッセージがメッセージ５１２のチャネルを通じて到着すると、結果として、読み出しトークン状態１５０１への遷移１５６１が起こる。

【０１０９】アプリケーションプロセス１３０９が複製ファイルに対する書き込み操作を試みた場合、結果として遷移１５５４が起こる。この遷移において、トークンサーバ１３１１はAPgetWtokenTS（斜体）メッセージに応答して、TSgetWtokenTS（ブロック体）メッセージを他方のトークンサーバ１３１１へ送り、その結果、状態１５０４に入る。次に、トークンサーバは、状態１５０４において、他方のトークンサーバ１３１１からのTSgrantWtokenRP_TS（斜体）メッセージを待機する。このメッセージが送られるチャネルは、バックエンドサーバ５１５へのメッセージ５１２のためのものである。TSgrantWtokenRP_TS（斜体）メッセージが到着すると、状態１５０３への遷移１５４３が起こる。

【０１１０】［同期の実装：図１、図１６、図１７］好ましい実施の形態では、同期は、複製ファイル１３２５の各ローカルコピーに対する７個のロックのセットによって実装される。図１６に、ロックの種類、ロックを有しなければならないシステム１３０１の要素、および、ロックの意味のリストである。ロック１６０１、１６０３、および１６０５は、それぞれ、プロセス１３０９が複製ファイル１３２５のローカルコピーに対してアトミック読み出し操作、順次読み出し操作、および書き込み操作を実行するために有しなければならないロックである。ロック１６０７は、ローカルバックエンドサーバ５１５が複製ファイル１３２５のローカルコピーに書き込みをするために有しなければならないロックである。ロック１６０９、１６１１、および１６１３は、現在ローカルホスト１３０２にある複製ファイル１３２５に対するトークンによって要求されるように、ローカルトークンサーバ１３１１によって要求される。例えば、ローカルホスト１３０２が書き込みトークンを有し、その書き込みトークンに対する要求を受け取った場合、ローカルトークンサーバ１３１１は、書き込みトークンロック１

６０９を解放し、トークンなしロック１６１３を獲得する。

【０１１１】図１７に、ロックの意味規則を示す。図１６の各ロックに対する行および列がある。行と列の交点にｘがある場合、相異なる要求者がその行のロックとその列のロックとを同時に保有することはできない。例えば、トークンサーバ１３１１がトークンなしロック１６１３を保有している場合、ローカルホスト１３０２にトークンがないという状況に対して要求されるとおり、プロセス１３０９は、順次読み出しロック１６０３またはローカル書き込みロック１６０５を有することはない。

【０１１２】好ましい実施の形態では、図１６のロックは、ＵＮＩＸオペレーティングシステムのSunOSオペレーティングシステムまたはSystem V Release 4のようなオペレーティングシステムによって提供される共有ロックおよび排他ロックにより実装される（SunOSはSun Microsystems, Inc.の商標である）。オペレーティングシステムによって提供されるロックにより、プロセスは、ファイルのバイトに対する共有ロックまたは排他ロックを取得することができる。さらに、このバイトは、他のファイルに対するロックを表すために使用することも可能である。こうして、好ましい実施の形態では、図１６のロックは、トークンファイル１３２７において、トークンファイル１３２７内のバイト列を各複製ファイル１３２５に割り当て、複製ファイルのバイト列内のバイトを使用して複製ファイルのロックを表現することによって実装される。複製ファイル１３２５に対するバイト列はスロットと呼ばれる。好ましい実施の形態では、各スロットは3バイトを有する。図１６の各ロックはスロットのバイトに対するロックの組合せによって表現される。

【０１１３】図1に、好ましい実施の形態において使用される組合せを示す。第1列は、システム１３０１内の複製ファイル１３２５のローカルコピーに対して使用されるロックのリストである。第2列は、どの種類のオペレーティングシステムロックが使用されるかを示す。Ｒは共有ロック、Wは排他ロック、およびＮＬはロックなしを示す。残りの列は、スロットのバイトを示す。バイトごとの列内のダッシュは、第2列で指定されるＯＳロックがそのバイトに対して獲得されていることを示す。こうして、プロセス１３０９は複製ファイル１３２５のローカルコピーに対するアトミック読み出しロック１６０１を獲得しているとき、オペレーティングシステムは複製ファイル１３２５に対するスロットのバイト0に共有ロックを有する。同様に、プロセス１３０９がローカルコピーに対する書き込みロック１６０５を獲得しているとき、オペレーティングシステムはスロットのバイト1および2に共有ロックを有し、バイト0に排他ロックを有する。

【０１１４】ＯＳロックのこのマッピングは、システム

１３０１のロックに対する図１７の衝突テーブルを実現する。衝突しているシステム１３０１のロックは、衝突しているＯＳロックおよび重複したオフセットにマップされ、一方、衝突していないシステム１３０１のロックは、衝突していないＯＳロックまたは重複しないオフセットにマップされる。好ましい実施の形態では、このマッピングはＳＬＥＶＥと呼ばれる同期ツールによって自動的に生成される。（エイ．スカーラ(A. Skarra)、「ＳＬＥＶＥ：イベント同期のための意味規則ロック(SLEVE: Semantic Locking for EVEnt synchronization)」、Proceedings of Ninth International Conference on Data Engineering（１９９３年）参照。）

【０１１５】［複製ファイルに対する高水準操作の実行］複製ファイルの各ローカルコピーが同様のすべての他のコピーと等価であるということの重要な結果として、上記の書き込み操作と全く同様にして高水準の操作を扱うことができる。書き込み操作では、書き込みトークン１３２７を有するホスト１３０２が複製ファイル１３２５のローカルコピーに対する書き込みを実行し、その後、その書き込みおよび書き込まれるデータを指定するメッセージを他のホスト１３０２へ送り、そこで、バックエンドサーバ５１５が、メッセージに指定された書き込み操作を、当該他のホスト１３０２内の複製ファイル１３２５のローカルコピーに対して実行する。全く同じことを高水準操作、例えば、２つの複製ファイルに関するソート（整列）・マージ（併合）を行う場合にも行うことができる。複製ファイルはすべてのホスト１３０２において等価であるため、書き込みトークン１３２７を有するホスト１３０２は以下のように進むことが可能である。ソート・マージを行いその操作に伴うすべての書き込みに対する書き込みバックアップメッセージ５１２を送る代わりに、ホスト１３０２は、ローカルコピーに対してソート・マージを実行してから、そのソート・マージ操作を指定するメッセージ５１２を他のホスト１３０２へ送ることが可能である。複製ファイル１３２５は他のすべてのホスト１３０２上で等価であるため、この指定されたソート・マージ操作の結果はすべてのホスト１３０２上で同一となる。このアプローチの利点は、ソート・マージ操作の指定を送ることは、書き込みトークンを有するホスト１３０２上のソート・マージの結果生じるすべての書き込み操作を他のホスト１３０２へ送るよりもずっと少ない時間およびネットワーク資源しか必要としないことである。

【０１１６】さらに詳細に説明すると、高水準操作は以下のような状況で使用することができる。各ホスト１３０２が２つの複製ファイル、すなわち、ソートされたマスタリストファイルおよび更新ファイルを有する。マスタリストファイルへの更新は更新ファイルに対して行われ、ソート・マージ操作が更新ファイルおよびマスタリストファイルに対して周期的に実行されて新たなマスタ

リストファイルが生成される。その後更新ファイルは削除され、再作成されて、このサイクルが再開される。更新ファイルに対する更新は、上記のような書き込み操作を使用して行われる。従って、ソート・マージ操作の時刻が来ると、すべてのホスト１３０２は同一の更新ファイルおよびマスタリストファイルを有する。書き込みトークンを有するホスト１３０２はそれ自身ソート・マージ操作を実行してから、書き込みバックアップメッセージ５１２を送る場合と全く同様にしてソート・マージ操作の指定を有するメッセージ５１２を他の各ホスト１３０２へ送る。この指定は、ソート・マージ操作のコードであることも可能であるが、この操作は反復して実行されるため、この指定は一般には、ソート・マージが更新ファイルおよびマスタリストファイルに対して実行されることを指定するコマンドラインである。次に、ソート・マージが他の各ホスト１３０２に対して実行される。同様に進行して、書き込みトークンを有するホスト１３０２は更新ファイルを削除し再作成して、同じ操作を指定するメッセージを他のホスト１３０２へ送る。もちろん、操作の指定は任意のレベルのものが可能である。例えば、上記の例では、更新ファイルのソート・マージならびに削除および再作成をすべてシェルスクリプトにおいて指定することが可能であり、操作指定は、そのシェルスクリプトとするか、または、他のすべてのホスト１３０２がそのシェルスクリプトのコピーを有する場合には、そのシェルスクリプトを呼び出すために使用するコマンドラインとすることも可能である。

【０１１７】［複製ファイルへのアクセスのトランザクション的同期］説明したように、複製ファイルシステムは、ファイルへの単一のアクセスに対する複製ファイル間の一貫性を保証する。また、複製ファイルシステムは、ファイルへのトランザクション的アクセスに対する一貫性を保証するためにも使用することができる。トランザクションとは、単一アクセスのシーケンスからなる単一の論理アクションである。トランザクションの例は以下の通りである。ファイルｆ内の各レコードｒは、プログラムＰがｒを読み出した回数を記憶する属性readnumを含むと仮定する。Ｐは以下の擬似コードにおいてreadnumをインクリメントする。ただし、関数のパラメータリストは省略してある。

【０１１８】for every r in f
read(f); increment_readnum(); write(f);

【０１１９】Ｐのいくつかのインスタンスが並行して実行されるときには、アクセスレベルの同期だけではｆのレコードにおける一貫性を保証するのに十分ではない。あるＰがｒを読み出した直後は、Ｐがreadnumをインクリメントしｒをｆに書き込むまでは、他のプログラムはｒの読み出しまたは書き込みを行うべきではない。読み出し－インクリメント－書き込みのシーケンスは、アトミック単位として同期すべきｒに対する単一の論理アクション（すなわち、トランザクション）である。

【０１２０】別個のトランザクション機構がない場合、プログラマは、オペレーティングシステムのロックプリミティブを用いてこのようなシーケンスに対するトランザクション同期の孤立性を実装することができる。プログラマは単にシーケンスをexclusive_lock(f)...unlock(f)のようなプリミティブでくくるだけである。一般の場合、計算は複数のファイルに関係し、プログラムは、孤立性を保証するときに複数のロックを要求しなければならない。例えば、計算が２つのファイルｆ１およびｆ２の内容に依存し、一方、その結果は他のファイルｆ３に記憶される。

read(f1); read(f2); computation(); write(f3);

そして、プログラムは以下のようにこのシーケンスをロック要求で囲む。

share_lock(f1); share_lock(f2); exclusive_lock(f3) ...
unlock(f3); unlock(f2); unlock(f1);

しかし、プロセスが一時に複数のロックを要求すると、デッドロックが起こる可能性がある。同期プロトコルを、トランザクション機構によって与えるか、または、オペレーティングシステムロックを使用するアプリケーションによって定義するかによって、デッドロックを防止するかまたは検出して解決するかしなければならない。

【０１２１】既存のオペレーティングシステムはローカルエリアネットワーク内の（リモート）ファイルをロックするプリミティブをサポートしている。しかし、分散環境では、高い通信コストのためにロックは効果であり、特にデッドロック検出は大域的アルゴリズムを必要とするため、より困難である。現在利用可能なオペレーティングシステムのロックプリミティブは、分散環境におけるトランザクション的同期を十分にサポートしていない。

【０１２２】ここで説明した複製ファイルシステムは、ローカルエリアネットワークまたは広域ネットワークにわたる複製ファイルにアクセスするプログラムシーケンスのデッドロックのないトランザクション的同期をサポートするTXというサービスを提供する。複製ファイルシステムは、並列プロセスを同期させるために必要なメッセージの数を最小にするプロトコルを実現する。

【０１２３】好ましい実施の形態では、複製ファイルシステムは、管理する各ファイルｆごとに現シーケンス番号（curSeqNf）を生成する。複製ファイルシステムは、プロセスが複製ツリー５０５にｆを作成するとき（またはこの複製ツリー５０５にｆを移動するとき）にcurSeqNfを０に初期化し、プロセスがwrite()操作でｆを変更するときcurSeqNfをインクリメントする。プロセスＰは、ｆに対する読み出しトークンまたは書き込みトークンを獲得することの一部としてcurSeqNfを受け取る。こ

の値は大域的（グローバル）である。すなわち、curSeqNfは、ｆへの変更に係るプロセスがＰに対してローカルであるかリモートであるかにかかわらず、そのすべての変更を反映する。

【０１２４】TXを使用するために、プロセスＰはフラグO_TXで複製ファイルｆをオープンする。Ｐが最初にｆの読み出しまたは書き込みをするとき、TXはcurSeqNfを局所変数locSeqNfに保存する。Ｐが次にｆの読み出しまたは書き込みをするとき、TXはlocSeqNfをcurSeqNfと比較する。これらが等しくない場合、その間に他のプロセスがｆを変更したことになり、TXはエラーを返す。そうでない場合、TXはオペレーティングシステムのreadまたはwriteファンクションを呼び出し、その結果を返す。

【０１２５】複製ファイルシステムは、デッドロックを避けるために、複製するファイル全体の順序を定義する。TXファンクションが複数のトークンを要求するとき、事前に定義された順序に従ってその要求を行う。

【０１２６】［インタフェース］TXの制御構造は、ＵＮＩＸインタフェースopen/close/read/writeの名前および返値を保ったまま再定義し拡張する関数インタフェース内にカプセル化される。TXは、errnoに対する新たな値としてETXを定義する。これは、トランザクション的同期によるエラーを意味する。また、TXは２つの新たな関数readtx()およびwritetx()を定義する。これらは、ファイルの集合にアクセスし、型fd_set*（システムコールselectの場合と同様）およびtx_t*のパラメータを有する。ただし、struct txt {char *buf;　int nbyte;}である。プログラムは、読み出しまたは書き込みをするファイルに対してfd_setにセットされたビットにより関数を呼び出し、ETXエラー復帰の場合、関数は、fd_setにおいて、curSeqNが変化しているファイル以外のすべてのビットをクリアする。インタフェースおよび擬似コードは以下の通りである。

【０１２７】・open(char *path; int flags; mode_t mode)
flagsがO_TXを含む場合、局所変数locSeqNpathを０に初期化し、O_TXをflagsから削除する。システムコールopenを呼び出し、その結果を返す。

【０１２８】・read(int fd; char *buf; int nbyte)
fdでオープンしたファイルｆに対する読み出しトークンを取得する。fdに対してO_TXがセットされていない場合、システムコールreadを呼び出し、読み出しトークンを解放し、復帰する。それ以外の場合、locSeqNf＝０であれば、curSeqNfをlocSeqNfに代入する。locSeqNf≠curSeqNfの場合、errno = ETXとセットし、そうでない場合、システムコールreadを呼び出す。読み出しトークンを解放し復帰する。

【０１２９】・write(int fd; char *buf; int nbyte)
fdでオープンしたファイルｆに対する書き込みトークンを取得する。fdに対してO_TXがセットされていない場合、ｆのローカルコピーに対するシステムコールwriteを呼び出し、それが成功した場合、その更新をリモートコピーへ送り、書き込みトークンを解放し、復帰する。それ以外の場合、locSeqNf＝０であれば、curSeqNfをlocSeqNfに代入する。locSeqNf≠curSeqNfの場合、errno = ETXとセットし、そうでない場合、上記のようにｆのコピーを更新し、O_TXの場合、locSeqNf（およびcurSeqNf）をインクリメントする。書き込みトークンを解放し復帰する。

【０１３０】・readtx(fd_set *readfds; struct tx_t *txp)
セットされた各ビットreadfds[fd]に対して、複製ファイルシステムによって定義される順序で、fdでオープンされたファイルｆに対する読み出しトークンを取得し、fdに対してO_TXがセットされている場合、次のことを実行する。locSeqNf＝０の場合、ｆを_nullとマークし、それ以外の場合、locSeqNf≠curSeqNfであれば、ｆを変更ありとマークする。いずれかのｆが変更ありの場合、errno =ETXとセットし、*readfdsにおいて、変更されたファイルに対するビット以外のすべてのビットをクリアする。いずれのｆにも変更がない場合、各ファイルｆに対して、システムコールreadを呼び出し、ｆが_nullである場合にはcurSeqNfをlocSeqNfに代入する。すべてのトークンを解放し復帰する。この関数は、いずれかのファイルが変更ありの場合にはいずれのファイルも読み出さず、それ以外の場合にはすべてのファイルを読み出す。

【０１３１】・writetx(fd_set *depends_on, *writefds; struct tx_t *txp)
セットされた各ビットdepends_on[fd]またはwritefds[fd]に対して、それぞれ、複製ファイルシステムによって定義される順序で、fdでオープンされたファイルｆに対する読み出しトークンまたは書き込みトークンを取得し、fdに対してO_TXがセットされている場合、次のことを実行する。locSeqNf≠０かつ≠locSeqNf≠curSeqNfである場合、ｆを変更ありとマークする。いずれかのｆが変更ありの場合、errno = ETXとセットし、*depends_onまたは*writefdsにおいて、変更されたファイルに対するビット以外のすべてのビットをクリアする。いずれのｆにも変更がない場合、*writefds内の各fdに対してシステムコールwriteを呼び出す。これが成功した場合、ｆのリモートコピーへその更新を送り、O_TXがセットされている場合、locSeqNfをcurSeqNf＋１とセットする（そしてcurSeqNfをインクリメントする）。すべてのトークンを解放し復帰する。この関数は、いずれかのファイルが変更ありの場合には書き込みを実行せず、それ以外の場合にはすべての書き込みを実行する。

【０１３２】・resettx(fd_set *fds)
セットされた各ビットfds[fd]に対して、fdでオープンされているファイルｆに対するlocSeqNfを０に再初期化

41

する。

【０１３３】［使用法］TXはアプリケーションが定義する再試行プロトコルをサポートする。例示のために、新しい各レコードとの新しいトランザクションを開始する＊

```
if(fd = open(f, O_RDWR O_TX)) < 0
    exit;
FD_ZERO(&fdset);
FD_SET(fd, &fdset);
for every r in f
    resettx(&fdset);
    for (try = TRY_NUM; try > 0; try--) {
        if read(fd, buf, nbyte) < 0
            exit;
        increment_readnum();
        if write(fd, buf, nbyte) >= 0
            break;
        if errno != ETX
            exit;
        /* そうでなければリセットし再試行する。*/
        /* いずれか他のプロセスがread()以降にｆを変更した。*/
       resettx(&fdset);
    }
```

【０１３４】プログラムは、新たな読み出し－インクリメント－書き込みのシーケンスを開始するごとにresettx()でlocSeqNfを再初期化する。これは、このシーケンスが、ｆの（前の）状態に依存しない単一の論理アクションから構成されるためである。

【０１３５】これに対して、ファイルｆ１、ｆ２および

42

＊アクセスについての上記の例を想起すると、TXを使用する擬似コードは、ファイルｆ内の各レコードｒに対して以下のようになる。

ｆ３に関する第２の例に対してTXを使用する擬似コードは、readtx()またはwritetx()の失敗後に再試行するときにのみ、locSeqNf1もしくはlocSeqNf2またはその両方をリセットする。

【０１３６】

```
/* O_TXフラグとともに、fd1～fd3でｆ１～ｆ３をオープンする。*/
/* バッファでtx_t配列を初期化する。*/
FD_ZERO(&readset);
FD_ZERO(&writeset);
for (try = TRY_NUM; try > 0; try--) {
    FD_SET(fd1, &readset);
    FD_SET(fd2, &readset);
    if readtx(&readset, txp) < 0 {
        if errno != ETX
            exit;
        resettx(&readset);
        continue;
    }
    Computation();
    FD_SET(fd3, &writeset);
    if writetx(&readset, &writeset, txp) >= 0
        break;
    if errno != ETX
        exit;
    /* そうでなければリセットし再試行する。*/
    /* いずれか他のプロセスがread()以降にｆ１またはｆ２を変更した。*/
    resettx(&fdset);
```

}

【０１３７】プログラムが後でファイルに対してresettx()を呼び出さずにｆ１、ｆ２またはｆ３にアクセスする場合、他のプロセスがこの間にそのファイルを変更していれば、アクセスは失敗する。

【０１３８】［応用］TXは、データベースシステムで使用される楽観的な、タイムスタンプに基づくアルゴリズムに類似した一種のトランザクション的同期をサポートする。楽観的方式では、トランザクションはデータを読み出してタイムスタンプを記録し、計算を行い、データベース更新のリストを作成する。コミット点において、トランザクションは、その更新をタイムスタンプとともにデータベースシステムへ送る。更新とタイムスタンプの組合せがトランザクション同期の孤立性を満足する場合、データベースはそのトランザクションをコミットする。そうでない場合、トランザクションは中断する（そして再始動される）。

【０１３９】これに対して、悲観的な、ロックに基づく方式では、トランザクションＴは各読み出しまたは書き込みの前に、オブジェクトに対する許可（すなわち、ロック）を取得する。ロックは、他のトランザクションによるそのオブジェクトにおける変更を排除する。そのロック要求が他のトランザクションのロックと衝突した場合、Ｔはそれ以上の計算を行わず、単に、他のトランザクションがそのロックを解放するまで待機する。悲観的アプローチは、中断および再始動によって作業が失われることが少ないため、トランザクションがデータベースオブジェクトに対して集中的な長期間の計算を実行するようなアプリケーションに対しては良好である。

【０１４０】楽観的方式は、オペレーティングシステムの領域では重要な効果を有する。オペレーティングシステムは通常いくつかのアプリケーションに同時にサービスするため、オペレーティングシステムによって制御される資源への公平なアクセスを各アプリケーションに提供することに注意しなければならない。楽観的方式はロックを待たないため、アプリケーションが資源へのアクセスを拒否される可能性は小さい。

【０１４１】［結論］以上では、どのようにして、ユーザレベルのバックアップファイルシステムを改良して、複製ファイルを有する分散システムを生成するかについて説明した。複製ファイルのローカルコピーに対して実行される順次読み出し操作および書き込み操作は、ただ１つのコピーが存在するファイルに対する読み出し操作および書き込み操作と同じ意味規則を有する。この意味規則は、分散システムの要素上のトークンサーバによって管理される読み出しトークンおよび書き込みトークンを使用する分散同期システムによって実現される。好ましい実施の形態では、分散同期システムは、複製ファイルの各ローカルコピーに対する７個のロックによって実現される。さらにこれらのロックは、３バイトのベクトルに対する標準的なオペレーティングシステムロックを使用して実装される。

【０１４２】上記で説明したのは、発明者が現在知っているユーザレベルの複製ファイルシステムを有する分散システムを実現する最適な形態であるが、多くの変形例が可能である。特に、本発明の原理は、上記のユーザレベルバックアップファイルシステムとは関係のないシステムでも使用可能である。例えば、ここで開示した同期技術は、複製ファイルを指定すること、または、ファイルバックアップ操作を実行することのために使用される技術とはほとんど独立であり、実際、複製ファイルに対する操作を同期させること以外の目的で使用することも可能である。さらに、同期は、好ましい実施の形態で使用したロックプロトコル以外の方法でも実現可能である。

【０１４３】

【発明の効果】以上述べたごとく、本発明によれば、複製ファイルを保守するために必要な操作が、分散システムのユーザレベルで実装可能となる。その結果、本発明は、特殊なハードウェアや特殊なオペレーティングシステムを必要としない。好ましい実施の形態は、ユーザレベルのバックアップファイルシステムの変更として実装される。

【図面の簡単な説明】

【図１】好ましい実施の形態における同期を実現するために使用されるロックの実装の図である。

【図２】ライブラリがユーザプログラムに対するインタフェースを再定義する方法の概観を示す図である。

【図３】動的リンクライブラリがオペレーティングシステムインタフェースを再定義するために使用可能であることを示す図である。

【図４】動的リンクライブラリがユーザレベル名前空間を提供するために使用可能であることを示す図である。

【図５】動的リンクライブラリを使用したユーザレベルバックアップファイルシステムの概略図である。

【図６】動的リンクライブラリ内のルーチンの概略図である。

【図７】ユーザレベルバックアップファイルシステムの好ましい実施の形態の概略図である。

【図８】カーネルサーバ３０５（ａ）によって提供される名前空間とユーザレベル名前空間の間の関係を示す図である。

【図９】ユーザレベル名前空間とカーネルサーバ３０５（ｂ）によって提供される名前空間の間の関係を示す図である。

【図１０】フロントエンド複製ツリー５０５の詳細図である。

【図１１】バックエンドマップ５１７の詳細図である。

【図１２】複製ファイルに対する同期システムの一部の

45

擬似コードを示す図である。

【図１３】同期システムのトークン機構のブロック図である。

【図１４】好ましい実施の形態におけるｗｒｉｔｅシステムコールを置換するコールの擬似コードを示す図である。

【図１５】２要素システムを有する分散システムにおいて複製ファイルに対する操作の同期を示す状態図である。

【図１６】好ましい環境で使用されるロックのテーブルの図である。

【図１７】図１６のロックの意味規則のテーブルの図である。

【符号の説明】

２０１　ユーザプログラム
２０３　ファンクション呼出し
２０５　復帰
２０６　インタフェース
２０７　ライブラリルーチン
２０９　ファンクション呼出し
２１１　復帰
２１３　インタフェース
２１５　システムルーチン
３０１　システム１
３０５　カーネルサーバ
３０６　ユーザプロセス
３０７　ディスク
３０９　アプリケーションプログラム（ユーザプログラム）
３１１　コール
３１３　復帰
３１５　オペレーティングシステムライブラリ１
３１７　コール
３１９　復帰
３２１　オペレーティングシステムライブラリ２
３２３　副次的効果
４０１　システム
４０３　オペレーティングシステムライブラリ
４０５　ユーザレベル名前空間
４０９　ユーザプロセス
５０１　ユーザレベルバックアップファイルシステム
５０３　アプリケーションプロセス
５０５　フロントエンド複製ツリー（ＦＲＴ）
５０７　ｌｉｂ．３ｄライブラリ
５０９　アプリケーションプログラム
５１１　主システム

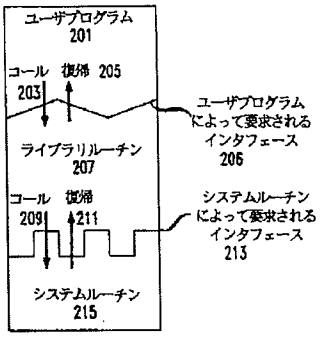
46

５１２　バックアップメッセージ
５１３　バックアップシステム
５１５　バックエンドサーバ
５１７　バックエンドマップ
６０１　ルーチン
６０３　ルーチン名
６０５　引数
７０１　ユーザレベルバックアップファイルシステム
７０３　ログファイル
７０９　パイプ
７１０　パイプ
７１１　パイププロセス
７１５　システムコールエンジン（ＳＹＳＣＡＬＬ　ＥＮＧ）
７１６　バックエンドログプロセス（ＢＬＰ）
７１７　モニタ
７１９　モニタ
８０１　名前空間
８０３　サブツリー
９０１　エントリ
９０３　ユーザレベル名前空間情報
９０５　バックアップシステム情報
１００３　ＭＡＸＴＲＹ
１００５　ＩＮＩＴ
１００９　ＲＰＬＯＰ配列
１０１３　ＲＴＲＥＥ　ＰＴＲ
１０１５　ＲＴＲＥＥ
１０２５　接続サーバ
１０２７　ファイルディスクリプタキャッシュ
１１０５　フロントエンドファイル識別子（ＦＦＩＤ）
１１０６　フロントエンドパス名（ＦＰ）
１１０７　フロントエンドプレフィクス（ＦＰＲ）
１１０８　サブツリーパス名
１１１１　バックエンドファイルディスクリプタ
１１１３　パス名マップ
１１１７　オープン複製ファイルリスト
１１１８　フロントエンドパス名
１１１９　バックエンドパス名
１３０１　分散システム
１３０２　ピアホスト
１３０７　トークンファイル
１３０９　プロセス
１３１１　トークンサーバ
１３２５　複製ファイル
１３２７　書き込みトークン
１３２８　読み出しトークン

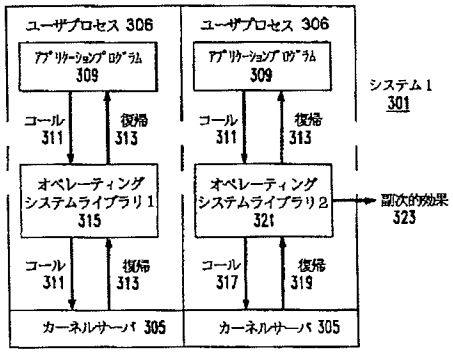
【図１】

システム1301ロック	OSロック	バイト 0	1	2
AR	R	—		
SR	R	—	—	
LW	R		—	—
	W	—		
RW	W	—		
WT	NL			
RT	W			—
NT	W		—	

【図２】



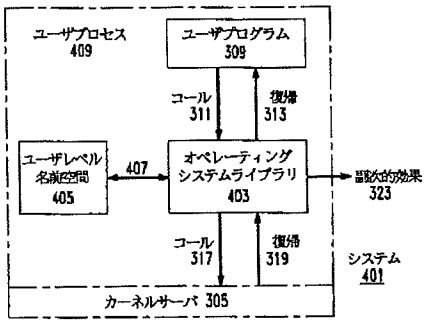
【図３】



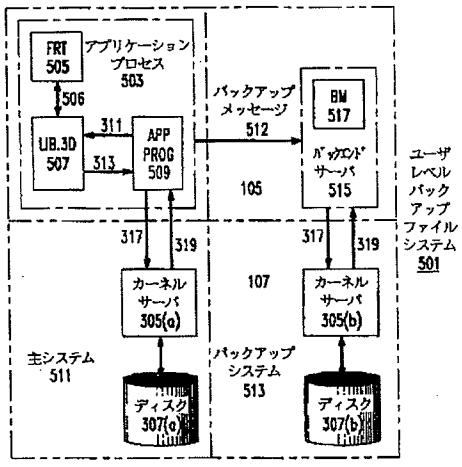
【図１７】

ロック	AR	SR	LW	RW	WT	RT	NT
AR			X	X			
SR			X	X			X
LW	X	X	X	X		X	X
RW	X	X	X	X			
WT					X		
RT			X			X	
NT		X	X				

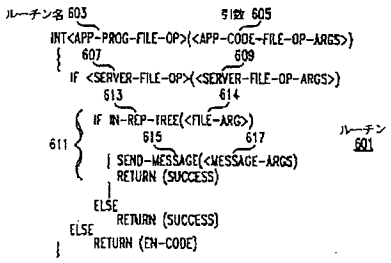
【図４】



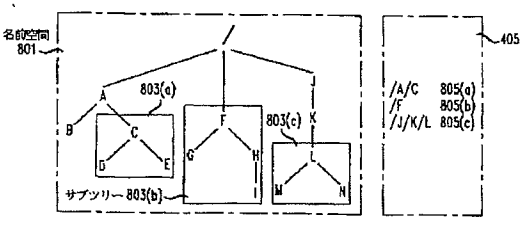
【図５】



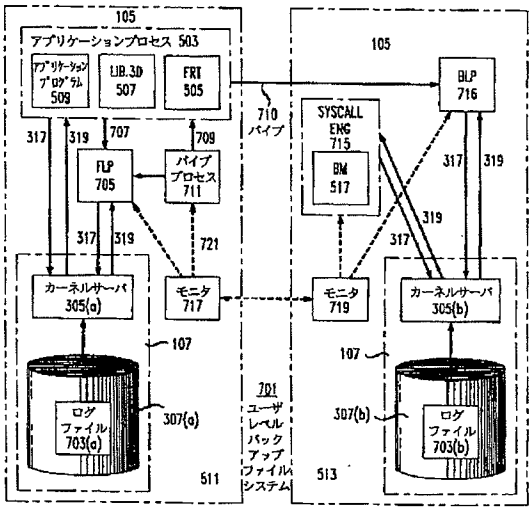
【図６】



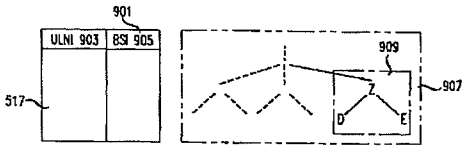
【図８】



【図７】



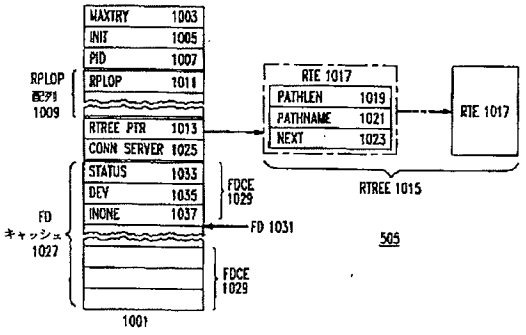
【図９】



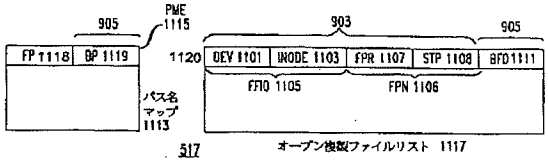
【図１２】

```
1   ssize_t write(int fildes, const void *buf, size_t nbyte)
2   {
3     get_write_token();          1201
4     r = syscall(SYS_write, fildes, buf, nbyte);
5     if (r > 0 && IsReplicatedTree(fildes))
6       send a message to the remote file system
7     release_token();
8     return (r);                 1203
9   }
```

【図１０】



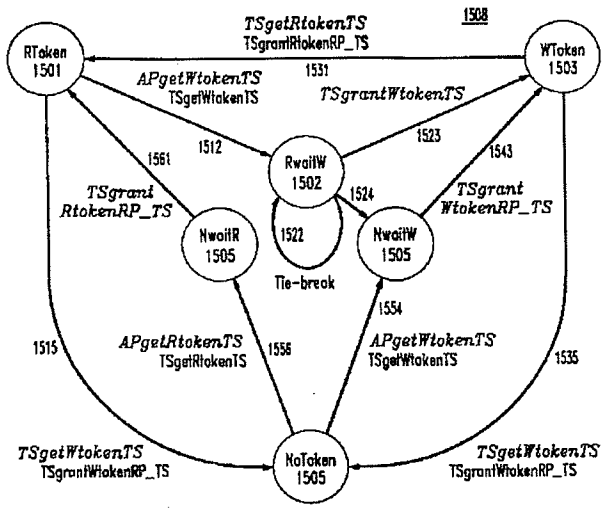
【図１１】



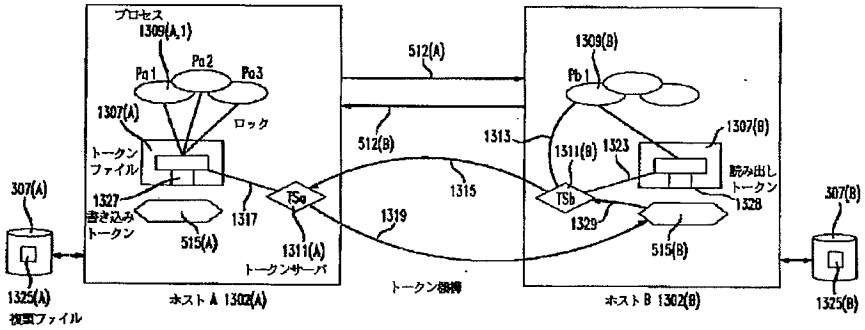
【図１４】

```
1   ssize_t write(int fildes, const void *buf, size_t nbyte)
2   {
3     TOKEN_REGION = fd2token(fildes);          1403
4     if (Lock(TOKEN_REGION.STATE, shared, non-block) == Fail)
5       send a request-token message to Token Server
6     Lock(TOKEN_REGION.STATE, shared, block);
7   Lock(TOKEN_REGION.TOKEN, exclusive, block);
8     r = syscall(SYS_write, fildes, buf, nbyte);
9     if (r > 0 && IsReplicatedTree(fildes))
10    send a message to the remote file system
11    UnLock(TOKEN_REGION);                     1401
12    return (r);
13  }
```

【図１５】



【図１３】



【図１６】

ロック	要求主体	意味
アトミック読み出し(AR)1601	プロセス 1309	ローカルコピーからのアトミック読み出し
順次読み出し(SR)1603	プロセス 1309	ローカルコピーからの順次読み出し
ローカル書き込み(LW)1605	プロセス 1309	ローカルコピーへの書き込みおよびリモートバックエンドサーバ 515 の待ち行列への送信
リモート書き込み(RW)1607	ローカルバックエンドサーバ 515	ローカルコピーへの書き込み
書き込みトークン(WT)1609	ローカルトークンサーバ 1311	ローカルトークンサーバは複製ファイルに対する書き込みトークンを有する
読み出しトークン(WT)1611	ローカルトークンサーバ 1311	ローカルトークンサーバは複製ファイルに対する読み出しトークンを有する
トークンなし(NT)1613	ローカルトークンサーバ 1311	ローカルトークンサーバは複製ファイルに対するトークンを有しない

フロントページの続き

(72)発明者　アンドレア　エイチ．スカーラ
アメリカ合衆国，07928　ニュージャージー，チャタム，オーチャード　ロード　26